

# **Protocol Audit Report**

Version 1.0

## **Protocol Audit Report**

Vesko.io

May 16, 2024

Prepared by: Vesko Lead Auditors: - Veselin VAchkov

#### **Table of Contents**

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
  - Scope
  - Roles
  - Issues found
- Findings
  - High
    - \* [H-1] Storing the password on-chain makes it visible to anyone, and no longer private
    - \* [H-2] PasswordStore::setPassword has no access controls, meaning a nonowner could change the password
  - Informational
    - \* [I-1] The PasswordStore::getPassword natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

### **Protocol Summary**

Protocol does X, Y, Z

#### Disclaimer

The Veselin's 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	Н	H/M	М
	Medium	H/M	М	M/L
	Low	М	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

#### 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.

#### **Issues found**

Severtity	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 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 called only by the owner of the contract.

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

**Impact:** Anyone can read the private password, severely 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 a locally running chain

```
1 make anvil
```

2. Deploy the contract to the chain

```
1 make deploy
```

3. Run the storage tool

```
1 cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

You can now parse that hex to a string with:

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 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.

# [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.

```
function setPassword(string memory newPassword) external {
    // @audit There are no access controls
    s_password = newPassword;
    emit SetNetPassword();
}
```

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

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

Code

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

**Recommended Mitigation:** Add a conditional to the setPassword function.

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

#### Informational

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

#### **Description:**

The PasswordStore: :getPassword function signature is getPassword() which the natspec says it should be getPassword(string).

**Impact:** The natspec is incorrect.

**Recommended Mitigation:** Remove the incorrect natspec line.

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