



# Protocol Audit Report

Version 1.0

*Cyfrin.io*

August 12, 2024

# Protocol Audit Report

ArtemiZ

August 8, 2024

Prepared by: Cyfrin Lead Auditors: - ArtemiZ

## Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
- High
- Medium
- Low
- Informational
- Gas

## Protocol Summary

A smart contract applicatoin for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

The ArtemiZ 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

Commit Hash:

```
1 7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
1 ./src/  
2 -- PasswordStore.sol
```

Roles

- Owner: The user who can set the password and read the password.
- Outsides: No one else should be able to set or read the password.

## Executive Summary

### 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 is no longer private

##### Description:

Variables stored in storage on-chain are visible to anyone, no matter the solidity visibility keyword meaning the password is not actually a private password. 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 can show 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 we use 1 because that is the storage slot of `s_password` in the contract.

```
1 cast storage <ADDRESS_HERE> 1
```

You will get an outout that looks like this: 0x6d7950617373776f726400000000000000000000000000000000

You can then parse the hex to a string with:

[illegible]

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 would also likely want to remove the view function as you would not want the user to accidentally send a transaction with the password that decrypts your password.

## [H-2] PasswordStore::setPassword has no access control, 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 // @audit any user can set a password
2 // missing access control
3 function setPassword(string memory newPassword) external {
4     s_password = newPassword;
5     emit SetNetPassword();
6 }
```

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

**Proof of Concept:** (Proof of Code)

Add the followeint to the `PasswordStore.t.sol` test file.

Code

```
1     function test_anyone_can_set_password(address randomAddress) public
2     {
3         vm.assume(randomAddress != owner);
4         vm.prank(randomAddress);
5         string memory expectedPassword = "myNewPassword";
6         passwordStore.setPassword(expectedPassword);
7
8         vm.prank(owner);
9
10        string memory actualPassword = passwordStore.getPassword();
11        assertEq(actualPassword, expectedPassword);
12    }
```

**Recommended Mitigation:**

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

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

## Informational

**[I-1] PasswordStore::getPassword natspec indicates a parameter that does not exist, causing the natspec to be incorrect**

**Description:**

```
1      // @audit their is no newPassowrd parameter!
2      * @param newPassword The new password to set.
3      */
4
5      function getPassword() external view returns (string memory)
```

The `PasswordStore::getPassword` function signature is `getPassword()` which the natspec say 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.
```