Logo

Protocol Audit Report

Prepared by: smart

Prepared by: smart Lead Auditors:

- smart kelvin

Table of Contents

Protocol Summary

Protocol for password

Disclaimer

The smart auditors 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 findng described in this document correspond the folloeing hashes

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
In Scope:
./src/
└── PasswordStore.sol
```

Roles

-- Owner; the user who can set the password and read the password --outside: no one should be able to set or read the password

Executive Summary

```
 This audit has identified a critical vulnerability in the smart contract related
to the storage of passwords. The issue stems from the fact that the passwordstore:
```

```
is_password variable, which is intended to be private, is visible to anyone on the
blockchain. Consequently, this flaw allows unauthorized users to read the private
password, undermining the security and intended functionality of the protocol.
```

Issues found

```
1. Visibility of Sensitive Data
Description: The passwordstore: is_password variable is visible on-chain, allowing
anyone to read the password.
Impact: Unauthorized access to the private password, compromising protocol
security.
Recommendation: Encrypt the password before storing it on-chain and remove any
view functions that expose sensitive data.
2. Lack of Access Controls
Description: The setPassword function lacks proper access control.
Impact: Any external user can change the password, leading to potential security
breaches.
Recommendation: Implement access control modifiers like onlyOwner to restrict
access to authorized users only.
```

severity	number of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

```
Visibility of Sensitive Data

Description: The is_password variable is visible on-chain.
Impact: Allows anyone to read the private password, compromising security.
Recommendation: Encrypt the password before storing it on-chain and remove any
view functions that expose it.

Lack of Access Controls
Description: The setPassword function lacks proper access control.
Impact: Allows unauthorized users to change the password, leading to security
breaches.
Recommendation: Implement access control (e.g., onlyOwner) to restrict function
access to authorized users only.
```

[H-1] TITLE (Root Cause + Impact)

variable store in storage is visibile to anyone, password can be seen by anyone

Description: **

all the data on-chain is visible to anyone, and can be read directly from the blockchain. the `passwordstore:is_password` variable is intended to be private varaible and only accessed throught 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: 0r proof of code

the below test case shows how anyone could read the password directly from the blockchain `bash` ``` 1. make a local running chain // anvil 2. deploy the contract 3 run the contract

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

```
you wiii get an output like this:
   `0x6d7950617373776f726400000000000000000000000000000000000000000014`

    ypu can parse the password with this:
       ```
       cast parse-bytes32-string
0x6d7950617373776f726400000000000000000000000000000000000000000014
       ```
       you will get the password output

       ```
       mypassword
       ```
```

Recommended Mitigation

```
due to this, the overall architecture of the contract should be rethought.one
could encrpte the password on-chain.  However, remove the view function as you
woulnt want the user to accidentally send a transaction with the password taht
decrytp your password .
```

[H-2]

`passwordstore:: setPassword` has no access control, non owner could change the password

Description:

the `passwordstore:: setPassword` function is set to be `eternal` function, however, the netspec of the function and the overall purpose of the fuction of the smart contart is that `THis function allows thw owner to set a new password`

~~~~
```javascript
function setPassword(bytes32 newPassword) external {
  @>    // @audit there are no access controls
    password = newPassword;
    emit setPassword()
}
```
~~~~

Impact:

anyone could change password . this will break the fuction of the protocol

Proof of Concept:

add the following the the test file

▶ click to expand

~~~~
```javascript
  function test_anyone_can_set_password(address randomAddress)  public {
    // vm.assume(randomAddress != address(0));
    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);
}

```

</details>
~~~~

Recommended Mitigation:

add the access control control to the `setPassword` function

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

[I-1] TITLE (Root Cause + Impact)

the `passwordStore::getpassword` netspec indicate a parameter that doesnst exist causing the netspec to be incorrect

Description:

Impact:

```
the  netspec is incorrect
```

Proof of Concept:

Recommended Mitigation:

remove the netspec

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

High

Medium

Low

Informational

Gas