Protocol Audit Report

Version 1.0

nem0x001

May 8, 2024

Password Store Audit Report

nem0x001

May 7, 2024

Prepared by: nem0x001 Lead Auditors: - ME

Table of Contents

Protocol Summary

A Protocol Detecated for storing passwords and retreving it . Users should be able to store a password and then retrieve it later. Others should not be able to access the password. # Disclaimer

The nem0x001 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

The finding described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/
==>PasswordStore.sol
```

Roles

- owner: The user who can set and retrieve password.
- outsiders: No one else can set or retrieve a password.

Executive Summary

- I have spent 15 hours with my team reviewing the code base manually.

Issues found

Severity	Number of Issues found
High	2
Medium	0
Low	0
info	1
Total:	3

Findings

High

[H-1] Password is stored on-chain.

Description: This finding underscores a security flaw within the codebase. Specifically, it exposes that passwords are stored within a state variable on the blockchain, rendering them easily accessible to anyone with access to the blockchain's data. This vulnerability presents a substantial risk, potentially resulting in unauthorized access, data breaches, and malicious activities. Despite `PasswordStore::s_password` being private and accessible solely through the `PasswordStore::getPassword` function, the fact that the password is stored on-chain transforms it into public information.

Impact: Anyone can read user's stored password.which break the protocol functionallity.

Proof of Concept: The following POC shows that anyone can read from the PasswordStore::s_password

1-Create a locally running chain

```
 anvil
```

2-Deploy the contract to the chain which set PasswordStore::s_password=myPassword

```
make deploy
```

3-Run the storage tool

```
cast storage <contract address> 1
 cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1
```

4-password in bytes

```
0x6d7950617373776f7264000000000000000000000000000000000000000000014
```

5- parsing to string

```
cast parse-bytes32-string 0
    x6d7950617373776f7264000000000000000000000000000000000000000000014
output:myPassword
```

Recommended Mitigation: Due to this the whole archeticture needed to be rethought.there are possible scenarios to consider as : Hashing Instead of storing the password in plain text, use a cryptographic hash function like keccak256. This function creates a unique and irreversible string based on the password. Anyone can see the hash on the blockchain, but it's impossible to determine the original password from it.

[H-2] `PasswordStore::setPassword` Broken Access control .

Description: This finding exhibits a critical security flaw stemming from broken access control mechanisms within the PasswordStore Contract. Despite its intent to allow only the contract owner to set and retrieve passwords, the implementation fails to enforce this restriction effectively. The PasswordStore::setPassword function lacks proper access control checks, enabling any address to overwrite the stored password, contrary to the intended behavior. Consequently, unauthorized parties can tamper with sensitive information, compromising the confidentiality and integrity of the system. This vulnerability poses a significant risk of unauthorized access and data manipulation, highlighting the importance of robust access control measures in smart contract development. `js /** @notice This function allows only the owner to set a new password. * @param newPassword The new password to set.`

`*/function setPassword(string memory newPassword)external { //@audit -F: BrokenAccess Control - anyone can set the password s_password = newPassword; emit SetNetPassword(); }` **Impact:** Anyone can set/change the password not only the owner. which break the contract functionallity.

Proof of Concept: The following POC shows that any one can set password using `PasswordStore::setPassword`function.

- We will make a test that compare the pass of owner and pass of notowner
- use the following code in the test file `PasswordStore.t.sol`

```
    function testAnyOneCanSetPass(address randomAddress) public {
         vm.assume(randomAddress != owner);
         vm.prank(randomAddress);
         string memory expectedPassword = "myNewPassword";
         passwordStore.setPassword(expectedPassword);
         vm.startPrank(owner);
         string memory actualPassword = passwordStore.getPassword();
         assertEq(actualPassword, expectedPassword);
     }
```

Recommended Mitigation: To address this vulnerability, it is recommended to enhance the access control logic within the contract, ensuring that only the contract owner can set and retrieve passwords. This involves modifying the `PasswordStore::setPassword` function to include a check for the caller's identity.

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

Informational

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