Protocol Audit Report

Version 1.0

NarwhalGuard

January 28, 2024

Protocol Audit Report

NarwhalGuard

January 18, 2024

Prepared by: NarwhalGuard Lead Auditors:

- Ammar Robbani

Table of Contents

- Low
  - [L-1] The `PasswordStore` contract exhibits an initialization timeframe vulnerability, which means that there is a period between contract deployment and the explicit call to `PasswordStore::setPassword` during which the password remains in its default state.
- Informational
  - [I-1] The `PasswordStore::getPassword` function lacks the `newPassword` parameter, even though it is mentioned in the natspec documentation.

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 NarwhalGuard 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

Repository for the audit:

https://github.com/Cyfrin/3-passwordstore-audit

Commit Hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner: The user who can set the password and read the password.

For this contract, only the owner should be able to interact with the contract.

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	1
Info	1
Gas Optimizations	0
Total	4

Findings

High

[H-1] The confidentiality of the password stored on-chain as private variable is compromised, rendering the protocol's password storage mechanism ineffective.

Description: Despite being declared as private, the `PasswordStore::s_password` variable is observable by external entities. The term private merely signifies inaccessibility by other smart contracts.

Impact: Any actor can access and view the content of `PasswordStore::s_password`.

Proof of Concept:

1. Launch the local chain:

```
make anvil
```

2. In a separate terminal, deploy the `PasswordStore` contract locally:

```
make deploy
// == Return ==
//0: contract PasswordStore 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

3. Obtain the contract address and access storage with index 1 (the storage index corresponds to the variable order in the contract):

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1
// 0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Convert the result from hex to string:

```
cast parse-bytes32-string 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
//myPassword
```

Recommended Mitigation: Employ encrypted passwords for the contract. Pre-encrypt the password off-chain before setting the encrypted password in the contract. Avoid including encryption and/or decryption methods within the contract to prevent public visibility by others.

[H-2] The `PasswordStore::setPassword` function is susceptible to misuse, allowing any user to alter the password, contrary to the intended functionality specified in the natspec documentation.

Description: The `PasswordStore::setPassword` function is susceptible to misuse, allowing any user to alter the password, contrary to the intended functionality specified in the natspec documentation.

Function:

```
1      function setPassword(string memory newPassword) external {
2  @>      // there is no access control here
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

Impact: All users possess the ability to repeatedly modify the password without adhering to the intended ownership restriction.

Proof of Concept:

```
1      function test_non_owner_can_set_password() public {
2          // Owner set the password
3          vm.startPrank(owner);
4          string memory prevPassword = "myPrevPassword";
5          passwordStore.setPassword(prevPassword);
6          vm.stopPrank();
7
8          // The other set up the new password
9          address nonOwner = makeAddr("user");
10         vm.startPrank(nonOwner);
11         string memory newPassword = "myNewPassword";
12         passwordStore.setPassword(newPassword);
13         vm.stopPrank();
14
15         // The owner checks if the current password is the same as the
               previous one
16         vm.startPrank(owner);
17         string memory newActualPassword = passwordStore.getPassword();
18         assertNotEq(newActualPassword, prevPassword);
19         assertEq(newActualPassword, newPassword);
20         vm.stopPrank();
21     }
```

Recommended Mitigation: To address this issue, two effective mitigation methods are proposed:

1. Access control from OpenZeppelin library:
   Integrate the Ownable contract from the OpenZeppelin library into the `PasswordStore` con-

tract. This approach establishes a robust access control mechanism, allowing precise determination of users authorized to execute the `PasswordStore::setPassword` function. For further details on access control, refer to the OpenZeppelin documentation.

Example code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;

import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol"
    ;

contract PasswordStore is Ownable {
    constructor() Ownable(initialOwner) {
        initialOwner = msg.sender;
    }

    function setPassword() public onlyOwner {
        // only the owner can call specialThing()!
        ...
    }
}
```

<br/

2. Use validation for the owner:
   Implement a validation step within the `PasswordStore::setPassword` function to verify whether the caller is the designated owner of the contract. By incorporating this validation, the function will only permit password changes initiated by the owner, aligning with the ownership-based access control stipulated in the natspec documentation.

```
    function setPassword(string memory newPassword) external {
+       if (msg.sender != s_owner) {
+           revert PasswordStore__NotOwner();
+       }
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Low

[L-1] The `PasswordStore` contract exhibits an initialization timeframe vulnerability, which means that there is a period between contract deployment and the explicit call to `PasswordStore::setPassword` during which the password remains in its default state.

Description: The contract does not set the password during its construction (in the constructor). As a result, when the contract is initially deployed, the password remains uninitialized, taking on the default value for a string, which is an empty string.

During this initialization timeframe, the contract's password is effectively empty and can be considered a security gap.

Impact: The impact of this vulnerability is that during the initialization timeframe, the contract's password is left empty, potentially exposing the contract to unauthorized access or unintended behavior.

Recommended Mitigation: To mitigate the initialization timeframe vulnerability, consider setting a password value during the contract's deployment (in the constructor). This initial value can be passed in the constructor parameters.

Informational

[I-1] The `PasswordStore::getPassword` function lacks the `newPassword` parameter, even though it is mentioned in the natspec documentation.

Description: The `PasswordStore::getPassword` function lacks `newPassword` as an input parameter, contrary to the indication in the natspec documentation.

Function:

```
1     /*
2      * @notice This allows only the owner to retrieve the password.
3      * @param newPassword The new password to set.
4      */
5     function getPassword() external view returns (string memory) {
6         if (msg.sender != s_owner) {
7             revert PasswordStore__NotOwner();
8         }
9         return s_password;
10    }
```

Recommended Mitigation: Revise the natspec documentation to accurately reflect the absence of the `newPassword` parameter in the `PasswordStore::getPassword` function. This adjustment will prevent confusion among developers and ensure accurate understanding of the function's signature.