Protocol Audit Report

Version 1.0

Arie71

September 10, 2024

Protocol Audit Report

Arie71

September 6, 2024

Prepared by: Arie71 Lead Security Researcher: - Gabriel Dibiya

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - 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 non-owner can 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

PasswordStore is a protocol dedicated to storage and retrieval of a user's passwords. The protocol is designed to be used by a single user, and is not designed to be user by multiple users. Only the owner should be able to set and access this password.

Disclaimer

The Arie71 team made 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 findings described in this document correspond to the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner - Only the owner may set and retrieve their password
- Outsiders: No one else should be able to set or read the password.

Executive Summary

The audit focused on identifying critical security vulnerabilities and potential improvements within the PasswordStore smart contract. Three key findings were identified, two of which posed high-severity risks, while the third was an informational issue that, while not directly harmful, could lead to confusion or misuse. Below is a summary of each finding and the recommended mitigations

The audit revealed serious vulnerabilities in the PasswordStore contract that could severely compromise its functionality and user security. Immediate action is required to address the on-chain password exposure and the lack of access controls on the setPassword function. Fixing these issues is crucial to restoring the intended privacy and security of the contract. Additionally, improving documentation accuracy ensures better maintainability and clarity for future users or developers.

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 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 only called by the owner or the contract.

We show one such 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

```
anvil
```

2. Deploy the contract to the chain

```
forge script script/DeployPasswordStore.s.sol:DeployPasswordStore --rpc-url http://localhost:8545 --private-key <ANVIL_PRIVATE_KEY_HERE> --broadcast
```

3. Run the storage tool

We use 1 because that's the storage slot of `s_password` in the contract.

```
cast storage <CONTRACT_ADDRES_HERE> 1 --rpc-url http://localhost:8545
```

You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

And get an output of:

```
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 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 accidentially send a transaction with the password that decrypts y0ur password.

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner can 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      function setPassword(string memory newPassword) external {
2  @>      // @Audit - There are no access controls
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

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

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

Code

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

Recommended Mitigation: Add an access control conditional to the `setPassword` function.

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

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 accassed

through the `PasswordStore::getPassword` function which is intended to be only called by the owner or the contract.

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

Impact:

Proof of Concept:

Recommended Mitigation:

Informational

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

Description:

The NatSpec comment for the `PasswordStore::getPassword` function incorrectly documents a parameter (`newPassword`) that doesn't exist in the function signature. The function `getPassword` does not accept any parameters, but the NatSpec suggests it should take a `newPassword` parameter, leading to a mismatch between the documentation and the actual function.

```
    /*
     * @notice This allows only the owner to retrieve the password.
     * @param newPassword The new password to set.
    */

    function getPassword() external view returns (string memory) {
        // Function logic
    }
```

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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