PasswordStore Security Review

Version 1.0

Marco Besier

February 19, 2024

PasswordStore – Security Review

Marco Besier

February 19, 2024

Prepared by: Marco Besier

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 used by multiple users. Only the owner should be able to set and access the password.

Disclaimer

I make all effort to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security review is not an endorsement of the underlying business or product. The review was time-boxed and the code was solely reviewed with regards to the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	C	H	M
Likelihood	Medium	H	M	L
	Low	M	L	L

Audit Details

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

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

Roles

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

Issues found

Severity	Number of issues found
Critical	2
High	0
Medium	0
Low	0
Info	1
Total	3

Findings

Critical

[C-1] Storing the password on-chain exposes it to everyone.

Description All data stored on the blockchain is publicly visible and can be directly accessed. The `PasswordStore::s_password` variable is designed as a private variable, intended to be accessed solely through the `PasswordStore::getPassword` function, which should only be invoked by the contract's owner.

Impact The public visibility of the password severely compromises the protocol's security.

Proof of Concept The following Proof of Concept (PoC) demonstrates how the password can be directly read from the blockchain by anyone:

Step 1: Initialize a local blockchain.

```
make anvil
```

Step 2: Deploy the contract to this local blockchain.

```
make deploy
```

Step 3: Read the password from storage using `cast`.

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

Note: The number 1 is used because it is the storage slot of `PasswordStore::s_password`.

The output will be:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

This hexadecimal can be converted to a string via:

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

The output will be:

`myPassword`

Recommended Mitigation Considering this vulnerability, it's advisable to rethink the contract's overall architecture. One approach could be to encrypt the password off-chain before storing the encrypted version on-chain. This method would necessitate the user remembering an additional off-chain password to decrypt the on-chain password. Furthermore, it may be advisable to eliminate the view function to prevent the user from inadvertently broadcasting a transaction that includes the decryption password.

[C-2] Lack of access controls in `PasswordStore::setPassword` allows non-owners to change the password.

Description The `PasswordStore::setPassword` function is designed to be external, implying that it should be callable from outside the contract. However, according to the function's NatSpec documentation and its intended purpose, it should only allow the contract's owner to set a new password. Currently, the function lacks any form of access control to enforce this restriction.

```
    function setPassword(string memory newPassword) external {
@>      // @audit - There are no access controls.
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact The absence of access controls on the `PasswordStore::setPassword` function permits any user to alter the password, severely compromising the contract's security and its intended functionality.

Proof of Concept To demonstrate this vulnerability, insert the following test into the `PasswordStore.t.sol` file:

```
    function test_anyone_can_set_password(address randomAddress) public
        {
        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);
    }
```

Confirm the test's success with:

```
forge test --mt test_anyone_can_set_password
```

Recommended Mitigation Implement an access control check in the `PasswordStore::setPassword` function to ensure that only the owner can update the password.

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

Informational

[I-1] Discrepancy in `PasswordStore::getPassword` NatSpec and function signature.

Description The `PasswordStore::getPassword` function is defined with the signature `getPassword()`, implying it does not accept any parameters. However, the NatSpec documentation for this function incorrectly suggests a parameterized version `getPassword(string)` by mentioning a non-existent parameter.

```
    /*
     * @notice This allows only the owner to retrieve the password.
@>   * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

Impact The NatSpec documentation does not accurately reflect the function's implementation, potentially causing confusion.

Recommended Mitigation To align the documentation with the actual function implementation, the incorrect NatSpec line should be removed. The corrected documentation would then look as follows:

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