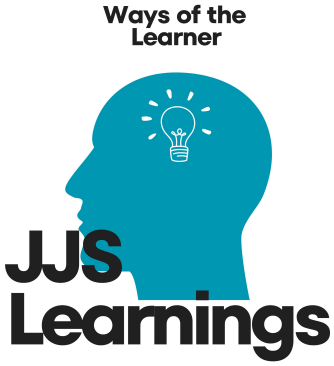
Protocol Audit Report

Jordan J. Solomon

May 21, 2024



Protocol Audit Report

Version 1.0

Jordan J. Solomon

July 4, 2024

Prepared by: Jordan J. Solomon

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
    - [H-1] Passwords Stored On-Chain Are Visible to Anyone, Regardless of Solidity Variable Visibility
    - [H-2] Missing Access Control for Setting Passwords: Any Address Can Set a New Password
  - Medium
  - Low
  - Informational
    - [I-1] The PasswordStore::getPassword natspac indicates a paramater the doesn't exist, causing the natspac to be incorrect
  - Gas

Protocol Summary

PasswordStore is a protocol designed to manage user passwords securely. It allows users to create, change, and retrieve their passwords while ensuring that only the user who created the password has access to it.

Disclaimer

The 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 findings described in this document correspond with 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.

Executive Summary

Description of how the audit went

Issues found

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

Findings

High

[H-1] Passwords Stored On-Chain Are Visible to Anyone, Regardless of Solidity Variable Visibility

Description: All data stored on-chain is accessible to anyone and can be directly read from the blockchain. The Password::s_password variable is intended to be private, accessible only through the Password::getPassword() function, which should only be invoked by the contract owner.

Impact: The password lacks privacy.

Proof of Concept: The following test demonstrates how anyone can read the password directly from the blockchain using Foundry's Cast tool, bypassing ownership requirements.

1. Start a locally running chain:

```
make anvil
```

2. Deploy the contract to the chain:

```
make deploy
```

3. 3. Use the Cast tool to inspect the contract's storage. Replace <ADDRESS_HERE> with the actual contract address.

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

The output will resemble:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Convert the hexadecimal output to a string:

```
cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000000
```

The result will be:

```
myPassword
```

Recommended Mitigation: Given the challenges of securing passwords on the blockchain, reconsidering the contract's architecture is advisable. Encrypting the password off-chain and storing the encrypted version on-chain is a viable approach. This requires users to remember an additional password off-chain for decryption purposes. Additionally, removing the view function could prevent accidental transactions that might expose or misuse the decryption password.

[H-2] Missing Access Control for Setting Passwords: Any Address Can Set a New Password

Description: The PasswordStore::setPassword function is marked as external, yet the NatSpec comment asserts that it should only be accessible by the owner. However, there are no checks to confirm the caller's identity.

Details

Code

```
    /*
     * @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 - There are no access controls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Without proper access controls, anyone can change the password, potentially locking out the legitimate owner.

Proof of Concept: A simple fuzz test demonstrates that non-owner addresses can set a new password.

Details

Test Code

```
function test_anyone_can_set_password(address randomaAddress) public {
        vm.assume(randomaAddress != owner);
        vm.prank(randomaAddress);
        string memory attackPassword = "attackPassword";
        passwordStore.setPassword(attackPassword);

        vm.prank(owner);
        assertEq(passwordStore.getPassword(), attackPassword);
    }
```

Recommended Mitigation: Implement access control logic to ensure that only the owner can call the PasswordStore::setPassword function.

Access Control Code Example

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

Medium

None ## Low None ## Informational ### [I-1] The PasswordStore::getPassword natspac indicates a paramater the doesn't exist, causing the natspac to be incorrect

Description: The PasswordStore::getPassword function signature is getPassword() while the natspac says it should be getPassword(string).

```
/*
     * @notice This allows only the owner to retrieve the password.
@>   * @param newPassword The new password to set.
     */
@>  function getPassword() external view returns (string memory)
    {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

Impact: The natspac is incorrect.

Recommended Mitigation: Remove the incorrect natspac line.

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

Gas

None