Prepared by: Florence Njeri

Table of Contents

Protocol Summary

The PasswordStore protocol is designed to let an owner store and update a private password on-chain and restrict retrieval to the owner

Disclaimer

Florence 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

m		**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 Scope Details

The findings described in this report correspong to the manual security review done on the following commit hash: `7d55682ddc4301a7b13ae9413095feffd9924566`

In Scope:

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

Solc Version: 0.8.18 **Chain(s) to deploy contract to**: Ethereum

Roles

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

Executive Summary

Issues found

Severity	Number of Issues Found
High	2
Medium	
Low	
Info	1
Total	3

Findings

High

[H-1] Variables stored on the blockchain are visible to anyone

Description: `private` only means that other contracts can't read the variable, but anyone with access to the blockchain can still read the value stored in a private variable. The `PasswordStore::s_password` variable is intended to be private and only accessible by the `PasswordStore::getPassword()` function, and only by the owner of the contract.

Impact: The password is visible to anyone and not secret, severely breaking the confidentiality and secrecy principle of the protocol.

Proof of Concept: The following test case proves that anyone can read the password:

1. Start a locally running chain by running: `make anvil`
2. Deploy the contract on the chain by running: `make deploy` in a separate terminal
3. Run the storage tool and target storage slot `1` for `s_password` (since it is the second variable declared in the contract):

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
http://127.0.0.1:8545
```

4. Run:

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

This outputs: `myPassword`

This proves that the password is stored on the chain and can be read by anyone.

```
┌──(lfnjeri㉿DESKTOP-572D2FT)-[/mnt/c/Users/Lydiah/Desktop/Practice/PasswordStore]
└─$ cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url http://127.0.0.1:8545
0x6d7950617373776f726400000000000000000000000000000000000000000014

┌──(lfnjeri㉿DESKTOP-572D2FT)-[/mnt/c/Users/Lydiah/Desktop/Practice/PasswordStore]
└─$ cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000014
myPassword
```

Recommended Mitigation: Rewrite the architecture for handling the password. Do not store secrets such as passwords, API keys, or encryption keys directly on-chain—even in private variables. Anything stored on the blockchain is inherently public and permanently visible to anyone who inspects the state.

Instead:

- Only store encrypted or hashed passwords on-chain.
- Use off-chain storage: Store sensitive data off-chain (e.g., in a secure backend service or database).

[H-1] `PasswordStore::setPassword()` has no access control checks, meaning anyone can change the password

Description: The `PasswordStore::setPassword()` function is `external` and contains no access control check. This means anyone can change the contract password, breaking the intended functionality that should only allow the owner to set the password.

```
/**
 * Access control issue >> anyone can set the password, not just the owner as
expected
 */
function setPassword(string memory newPassword) external {
```

```
    s_password = newPassword;
    emit SetNetPassword();
}
```

Impact: Anyone can set or change the password of the contract, breaking the contract's intended functionality.

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

```
function test_anyone_can_set_password(address randomAddress) public {
    // Use a random address to emulate a non-owner setting the password
    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);
}
```

When you run the test, the assertion passes, proving that anyone can set the password.

Recommended Mitigation: Add an access control condition to restrict who can set the password:

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

Informational

[L-1] Incorrect parameter documentation in `PasswordStore::getPassword()`

Description: The documentation mentions a parameter that doesn't exist: `@param newPassword The new password to set.` This makes the function documentation incorrect.

Impact: The NatSpec is inaccurate and misleading.

Recommended Mitigation: Remove the incorrect NatSpec line:

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