Protocol Audit Report

Version 1.0

Alex Langevin

February 21, 2024

PasswordStore Audit Report

Alex Langevin

February 19, 2024

Prepared by: Alex Langevin

Table of Contents

Protocol Summary

PasswordStore is a protocol that enables users to store and retrieve their passwords. The protocol is designed to be used on a per user basis. Only the contract's owner should be able to set and retrieve this password.

Disclaimer

Alex Langevin 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

I use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

- Commit Hash: 2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
- In Scope:

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

Roles

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

Executive Summary

A 2-hour audit revealed critical issues in the protocol. Storing unencrypted passwords on-chain poses a security risk. Access control and initialization concerns were identified. Two informational findings suggest improvements in adhering to Solidity best practices. Urgent action is recommended to address these issues and enhance protocol security.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	1
Informational	2

Findings

High

[H-01] Private variables are visible to anyone

Description:
Variables in storage are visible to anyone, even the ones marked **`private`** like `PasswordStore::s_password`. To read their value, all you need is the contract address and the storage slot in which the variable is stored.

Impact:
The protocol's main functionality is to enable an owner to safely save a password that will only be visible to him. Therefore, the protocol is rendered useless by the fact that all data stored on-chain is public.

Proof of Concept:

The below test proves that anyone can read `PasswordStore::s_password` from the blockchain.

1. Create a locally running chain

```
1     make anvil
```

2. Deploy the `PasswordStore::PasswordStore` contract to the local chain

```
1     make deploy
```

3. Using cast, read the value of `PasswordStore::s_password`

In the code below, 1 is the storage slot and `0x5FbDB2315678afecb367f032d93F642f64180aa3` is the contract address.

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

Using cast, let's decode the output of the previous command to a string.

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

The output is `myPassword` which is the correct password.

Recommended Mitigation:

All solutions will require a significant refactoring of the protocol. An option is to encrypt the password value before storing it on-chain.

[H-02] `PasswordStore::setPassword` has improper access control

Description: The documentation clearly states that `PasswordStore::setPassword` should only be callable by the owner. The current implementation has no form of access control which enables anyone to call the function and modify the value of `PasswordStore::s_password`.

Impact: Anyone can call `PasswordStore::setPassword` and modify `PasswordStore::s_password`. This breaks the protocol's main functionality since no password is secret.

Proof of Concept:

The following foundry test fails since the transaction does not revert as expected. `address(1)` is not the owner's address.

```
1     function test_non_owner_set_password(address attacker) public {
2         vm.assume(attacker != owner);
3
4         string memory newPassword = "enchiladas";
5
```

```
6           vm.prank(attacker);
7           passwordStore.setPassword(newPassword);
8
9           vm.prank(owner);
10          string memory password = passwordStore.getPassword();
11
12          assertEq(password, newPassword);
13      }
```

Recommended Mitigation:

The easiest way to fix the vulnerability is to implement an access control in `PasswordStore::setPassword` similar to the one in `PasswordStore::getPassword`.

```
1       function setPassword(string memory newPassword) external {
2           if(msg.sender != s_owner){
3               s_password = newPassword;
4               emit SetNetPassword();
5           }
6       }
```

Low

[L-01] Default password value at deployment

Description: During deployment, `PasswordStore::s_password` is not initialized. Therefore it takes the default string value of `""`. It only differs from this value once a first call is made to `PasswordStore::setPassword`. This means that for a period of time, all passwords will have the default value which could be a security vulnerability.

Impact: For the duration of time that the password is not initialized, the known default password can be used for unauthorized access.

Proof of Concept:

```
1       address owner = address(owner);
2
3       function test_default_password() public {
4           vm.startPrank(owner);
5           PasswordStore passwordStore = new PasswordStore();
6
7           string memory password = passwordStore.getPassword();
8           assertEq(password, "");
9       }
```

Recommended Mitigation:

Add a `string` argument in the `PasswordStore::constructor` that represents the initial password and initialize `PasswordStore::s_password` to its value.

```
    constructor(string memory initial_password) {
        s_owner = msg.sender;
        s_password = initial_password;
    }
```

Informational

[I-01] `PasswordStore::s_owner` should be an immutable

Description: Variables that are initialized at deployment and can't be changed should use the `immutable` modifier. This means that the value will be written in the contract's bytecode and instead of the storage. This is both safer and more efficient.

Impact: Using `immutable` makes the contract a bit safer and more efficient since there will not be a storage read operation.

Recommended Mitigation:

```
    address private immutable s_owner;
```

[I-02] Error in the comments in `PasswordStore`

Description: The `getPassword::PasswordStore` natspec lists an argument named `newPassword`. This is a copy-paste error from the `setPassword::PasswordStore`'s natspec.

Impact: The comment in the code contradicts the documentation of the protocol.

Recommended Mitigation:

Delete the following line.

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