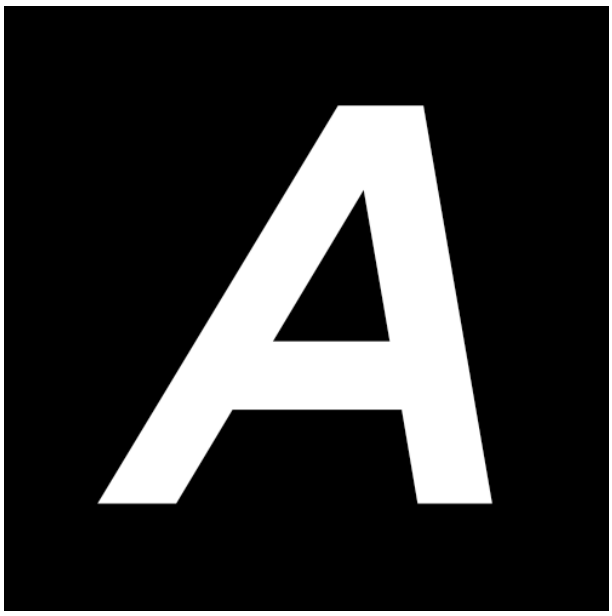

Protocol Audit Report

Version 1.0

beruf

January 1, 2024

Protocol Audit Report

beruf

January 1, 2024

Prepared by: beruf

Lead Security Researcher: - beruf

Table of Contents

Protocol Summary

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

Disclaimer

The beruf 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 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

Review went straight forward, finding some critial errors which serverly impact the protcol. We spent 2 hours with 1 security researcher using foundry.

Issues found

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

Findings

High

[H-1] Storing the password onchain makes it visible to anyone, and no longer private

Description: All data stored onchain is public and visible to anyone. The `PasswordStore::s_password` variable is intended to be hidden and only accessible by the owner through the `PasswordStore::getPassword()` function.

I show one such method of reading any data off chain below.

Impact: Anyone is able to read the private password, severly breaking the functionality of the protocol

Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

Create a locally running chain

```
make anvil
```

Deploy the contract to the chain

```
make deploy
```

Run the storage tool

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

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

You'll get an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to a string with:

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

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 encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

[H-2] `PasswordStore::setPassword()` has no access controls, a non-owner could change the password

Description: The `PasswordStore::setPassword()` function is `external` and has no access control. Anyone can call this function and change the password even the user isn't the owner. However

according to the functions's natspec `This function allows only the owner to set a new password.`

```
function setPassword(string memory newPassword) external {
    // @Audit - There are no Access Controls.
    s_password = newPassword;
    emit SetNewPassword();
}
```

Impact: Anyone can set the stored password, breaking the contracts intention that only the owner can set it.

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

Code

```
  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);
  }
```

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

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

Informational

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

Description:

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

The `PasswordSore::getPassword()` function signature is `getPassword()` which the natspec say it should be `getPassword(string newPassword)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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