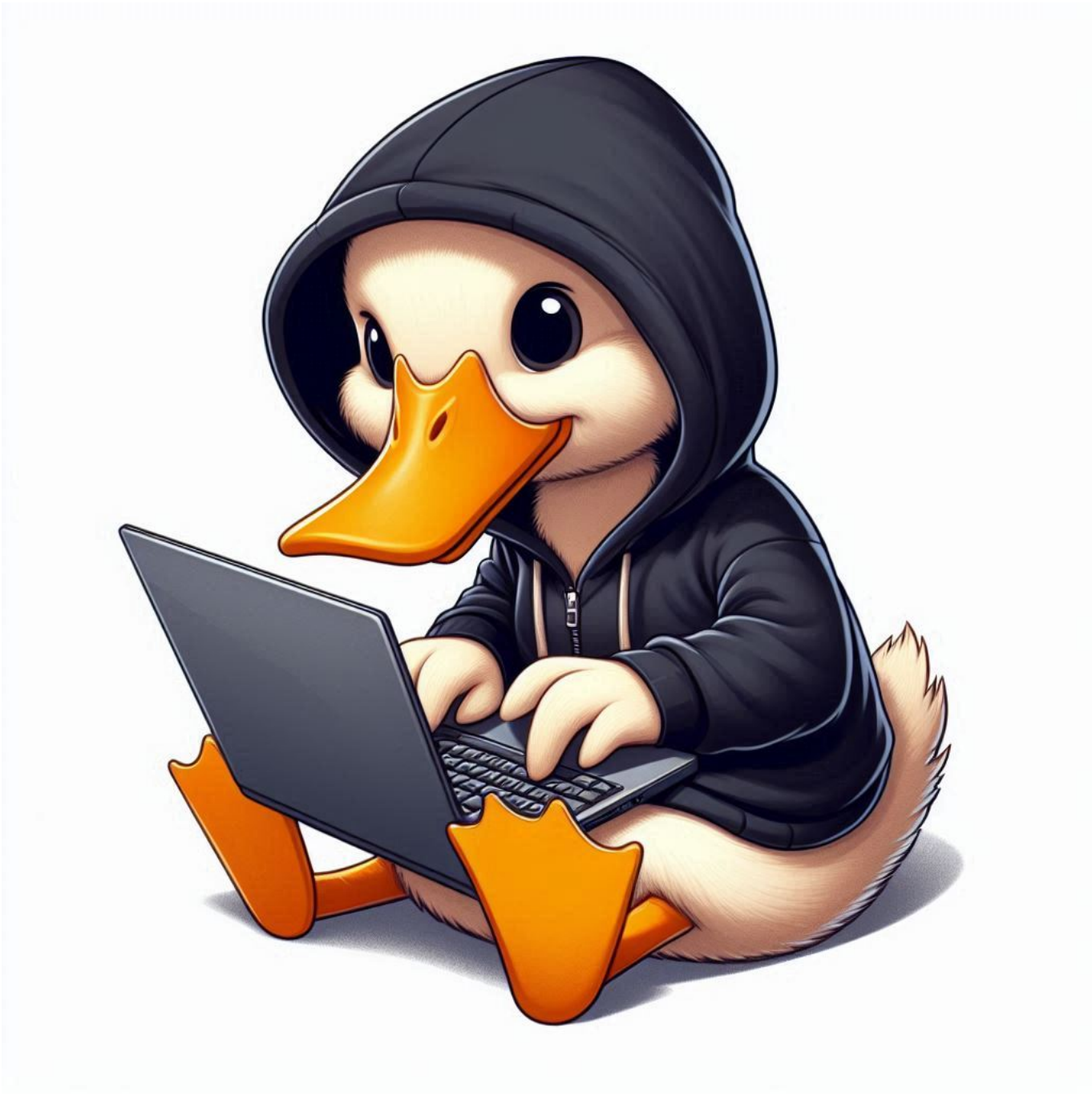
HackDuck



Protocol Audit Report: PasswordStore

Table of Contents

Protocol Summary

A smart contract application for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

The HackDuck 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

Scope

The PasswordStore.sol smart contract.

Roles

Executive Summary

Issues found

Findings

High

[H-1] Passwords stored on-chain are visable to anyone, not matter solidity variable visibility

Description: All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable, and only accessed through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

However, anyone can direclty read this using any number of off chain methodologies

Impact: The password is not private.

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.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

We use `1` because that's the storage slot of `s_password` 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
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

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 password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

[H-2] No access control on 'setPassword' funcion

Description: The `PasswordStore::setPassword` function should allow only the owner `PasswordStore::s_owner` to call it. Due to missing access control, it can be called by anybody, so anyone can set/change the password:

```
function setPassword(string memory newPassword) external {
@>      // @audit missing access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone calling the function `setPassword` can set the password.

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

```
 function test_non_owner_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(expectedPassword, actualPassword);
    }
```

Recommended Mitigation: Create and add a modifier for the function.

Example:

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

Medium

Low

Informational

[I-1] Incorrect natspec on `PasswordStore::getPassword`

Description The natspec of the function says the signature is `getPassword(string)` while it actually is `getPassword()`

Impact The natspec is incorrect

Recommended mitigation Remove the incorrect natspec line.

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

Gas