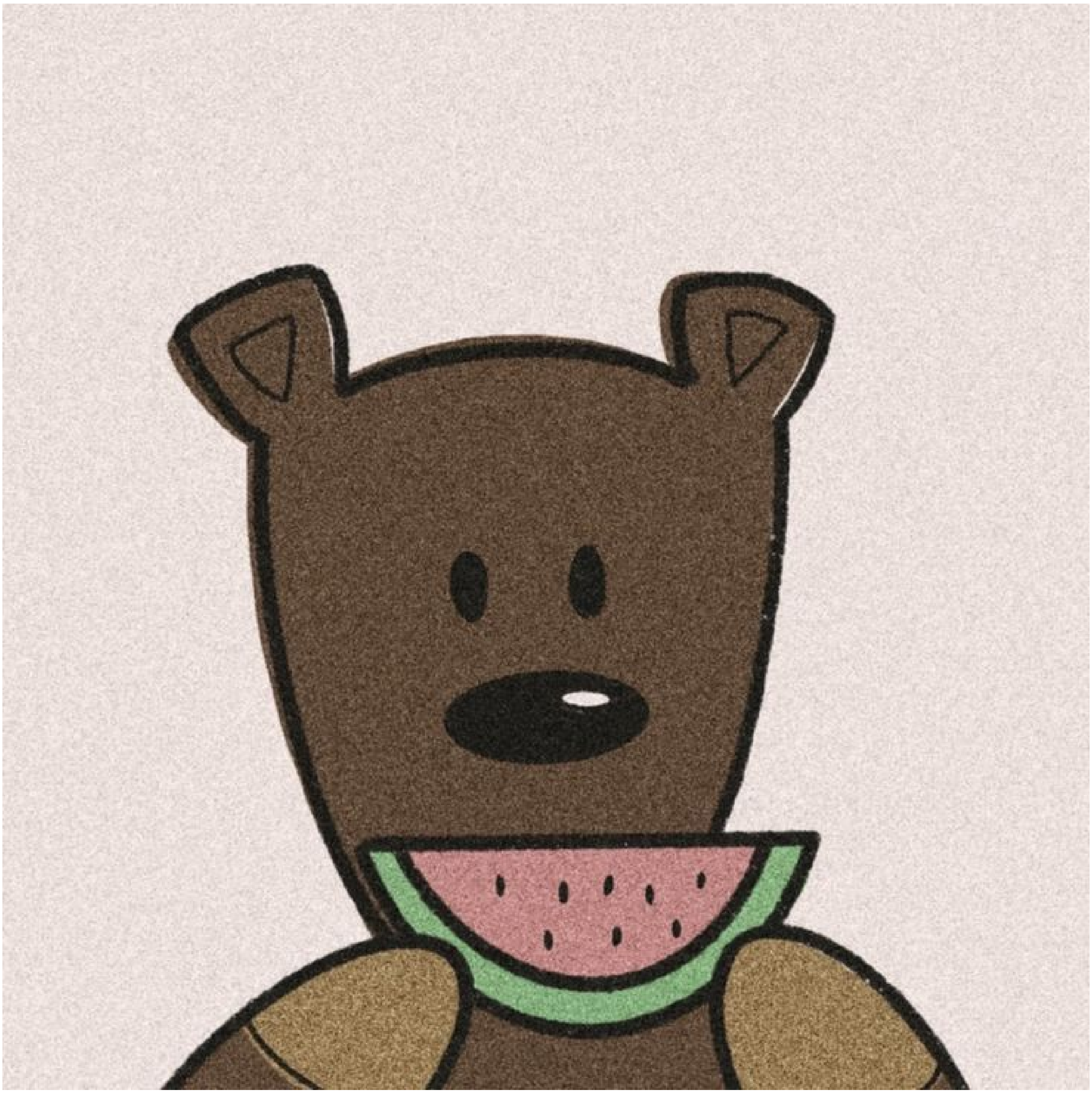

Protocol Audit Report

Prepared by: Uddercover

Table of Contents

Protocol Summary

Protocol allows a user to store and view a password. It doesn't allow an outsider to change or view a user's password.

Disclaimer

The security researcher, Uddercover, 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

- Github: https://github.com/Cyfrin/3-passwordstore-audit/tree/onboarded
- Commit Hash: 7d55682ddc4301a7b13ae9413095feffd9924566

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

Found some critical bugs and an informational one

Issues found

Severity	Number of Issues
Highs	2
Mediums	0
Lows	0
Gas	0
Informational	1
Total	3

Findings

High

[H-1] The password stored on-chain is visible to everyone, and not private at all

Description: The variable `PasswordStore::s_password` holds a password set by the owner, that should not be visible to anyone else. But the nature of the blockchain makes every piece of information on-chain publicly available, regardless of visibility specified. This means that the contents of the `PasswordStore::s_password` is visible to everyone, which is not the desired outcome.

An example of reading any data from the blockchain is given below in the proof of concept section.

Impact: The password is accessible to anyone, severely breaking the functionality of the protocol

Proof of Concept:

The below test shows how it's possible to read data with visibility set to `private` from the blockchain:

1. Run a local chain

```
make anvil
```

1. Deploy PasswordStore.sol to that chain

```
make deploy
```

3. Read the target storage slot of `s_password`, slot `1`

```
cast storage <PASSWORD_STORE_ADDRESS> 1
```

The above returns a hex value: `0x6d7950617373776f7264000000000000000000000000000000000000000000014`

4. Convert the hex value to a string using

```
cast --to-ascii <HEX_VALUE>
```

The above returns: `myPassword`

Recommended Mitigation: The protocol should reconsider the current project architecture and look into alternate methods for storing private data on-chain. The password could be encrypted off-chain and the encrypted data stored on-chain, but this introduces another set of challenges that would need to be addressed.

[H-2] Access control is not implemented in `PasswordStore::setPassword`, meaning that a non-owner can set password

Description: According the the natspec `@notice This function allows only the owner to set a new password`, the `PasswordStore::setPassword` function should allow only the owner to be able to call it but the logic restricting access to only the owner is not implemented. As such, anyone can call `setPassword` and set a new password.

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

Impact: Anyone can set/change the password of the contract, severely breaking the intended functionality.

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

▶ Code

```
    function test_non_owner_can_set_password(address random) public {
        vm.assume(random != owner);
        vm.prank(random);
        string memory expectedPassword = "MyNewPassword";
        passwordStore.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();

        assertEq(expectedPassword, actualPassword);

    }
```

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

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

Informational

[I-1] The `PasswordStore::getPassword` function natspec indicates a parameter that doesn't exist, this makes the natspec 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 `PasswordStore::getPassword` function signature is `getPassword()` but the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect

Recommended Mitigation: Remove the erroneous documentation

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