\begin{titlepage} \centering \begin{figure}[h] \centering \includegraphics[width=0.5\textwidth]{logo.pdf} \end{figure} \vspace*{2cm} {\Huge\bfseries Protocol Audit Report\par} \vspace{1cm} {\Large Version 1.0\par} \vspace{2cm} {\Large\itshape 0xhardhat\par} \vfill {\large \today\par} \end{titlepage}

\maketitle

Prepared by: 0xhardhat Lead Auditors:

- xxxxxxx

Table of Contents

Protocol Summary

This is a protocol designed to store and save password, its is designed for a single user and not for multiple users.Only owner should be able to store and access password.

Disclaimer

0xfoundry made all efforts 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

```
commit hash:2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

- solc version : 0.8.19
- chain(s) to deploy on : Ethereum

Roles

- Owner: Is the only one who should be able to set and access the password.
- For this contract, only the owner should be able to interact with the contract.

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing password varaible on-chain is makes it visible to anyone, and no loner private.

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

Impact: Anyone can read the private passwords which breaks the functionality of the protocol.

Proof Of Concept: The below test case showsn anyone can read the passwords directle from the blockchain.

1. Create a local blockchain

```
  make anvil
```

2. Deploy the contract to the chain

```
   make deploy
```

3. Run the storage

we use `1` because thats the storeg slot of `s_password` in the contract

we will get an output like this 0x6d7950617373776f72640000000000000000000000000000000000000000000014

then parse the hex back to string

```
  cast parse-bytes32-string
0x6d7950617373776f72640000000000000000000000000000000000000000000014

  Which will give an output of
```

mypassword

Recommended Mitigation: Consider revising the contract's architecture due to this requirement. One approach is to encrypt the password off-chain and then store the encrypted version on-chain. This method necessitates users remembering an additional off-chain password for decryption. It's important to remove the view function to prevent accidental transactions that might expose the decryption password.

Likelihood && Impact:

- Impact: HIGH
- Likelihod:HIGH
- Severity:HIGH

[H-2] `PasswordStore::setpassword` has no access control, a non-owner can change password.

Description: The `PasswordStore::setpassword` is set to be an external function ,The netspec and overall function and purpose of this smart contract `The function allows only the owner to set password`

```
    function setPassword(string memory newPassword) external {
     //@audi --there is no access control
          s_password = newPassword;
          emit SetNetPassword();
      }
```

Impact: Anyone can change/set the contract password which is severly breaking the contract intended purpose

Proof of concept: Add this to the `passwordStore.t.sol` test file.

▶ Details

```

      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 mitiation: Add a control access to the `setPassword`

▶ Details

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

Likelihood & Impact.

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

Informational

[I-1] The `PasswordStore::getPassword` function signature `getPassword()`, why the natspec say its should `getPassowrd(string)`.

Impact: natspec is in correct.

Recommended mitigation: Remove the incorrect natspce

```diff
-     * @param newpassword  The new password to set
```

Likelihood & Impact.

- Impact: NONE
- Likelihood: HIGH
- Severity: Information/Gas/Mon-crits