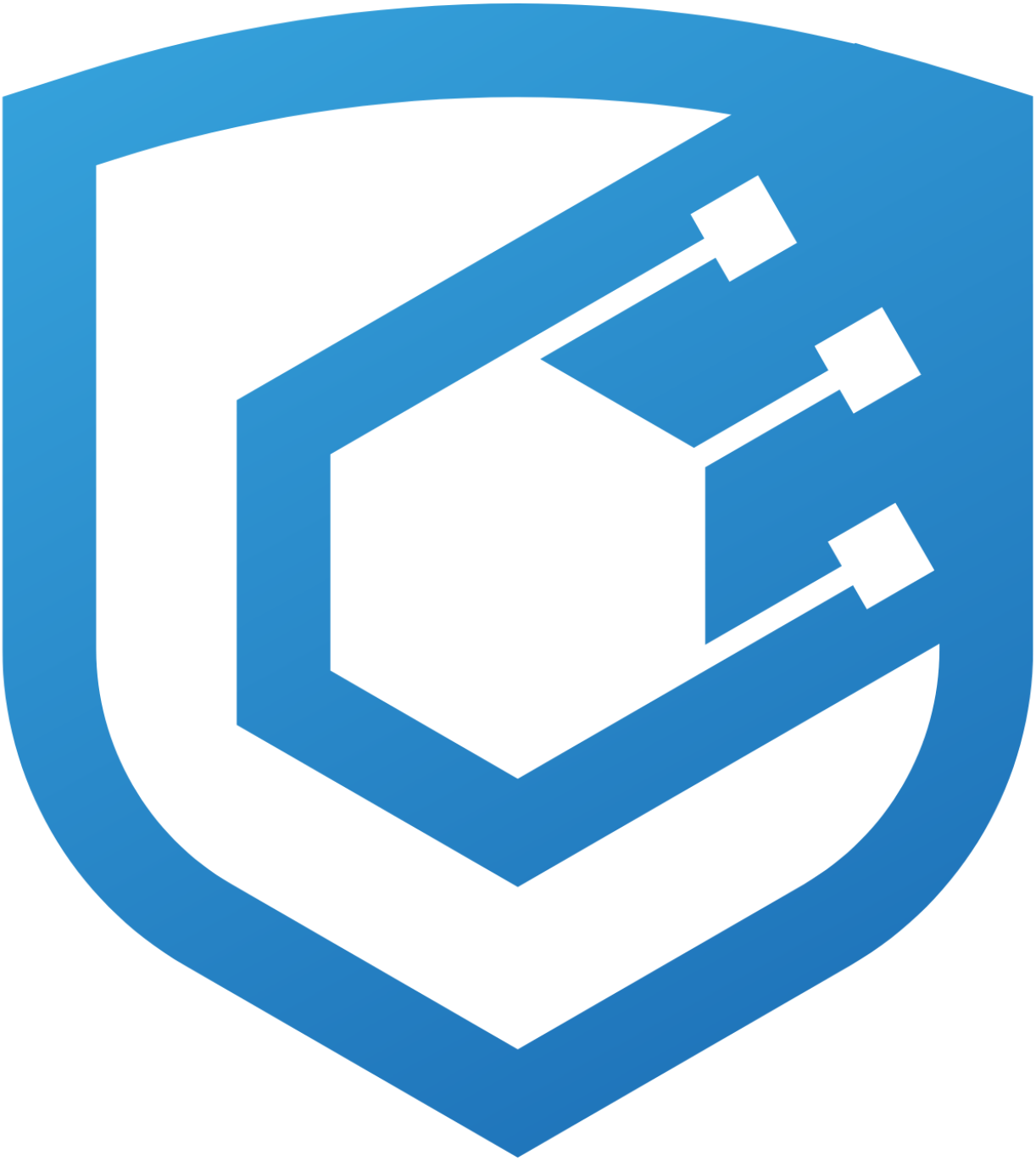

Passwordstore Audit Report

Prepared by: Jason

Table of Contents

Protocol Summary

A smart contract applicatoin 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 Jason 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:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

./src/ └── PasswordStore.sol

Roles

- Owner: The user who can set the password and the password.
- Outsiders: No one else should be able to set or read the password.

Executive Summary

Add some notes about how the audit went, types of things you found, etc.

We spent X hours with Z auditors using Y tools. etc

Issues found

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

Findings

High

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

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

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

Impact: Anyone can read the private password, severly breaking the functionality of the protocol.

Proof of Concept: (Proof of Code)

The below test case shows how anyone can read the password directly from the blockchain.

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-bytes-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 wouldnt want the user to accidently send a transaction with passwprd that decrypts your password.

Likelihood & Impact

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

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

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec of the function and overall purpose of the smart contract is that `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 SetNetPassword();
}
```

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

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);
}
```

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

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

Likelihood & Impact

- Impact: HIGH
- Likelihood: HIGH
- CRITICAL

- Severity: HIGH
-

Informational

The [I-1] `PasswordStrore::getPassword` natspec indicates a parameter that doesnt exist, causing the natspec to be incorrect

Description:

```
The `PasswordStore::getPassword` function signature is `getPassword` while the natspec says it should be `getPassword(string)`.
```

Impact: The natspec is incorrect

Recommended Mitigation: remove the incorrect natspec line.

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

Likelihood & Impact

- Impact: NONE
- Likelihood: HIGH
- Severity: Informational/Gas/Non-crits

informational: Hey, this isn't a bug, buut you should know...