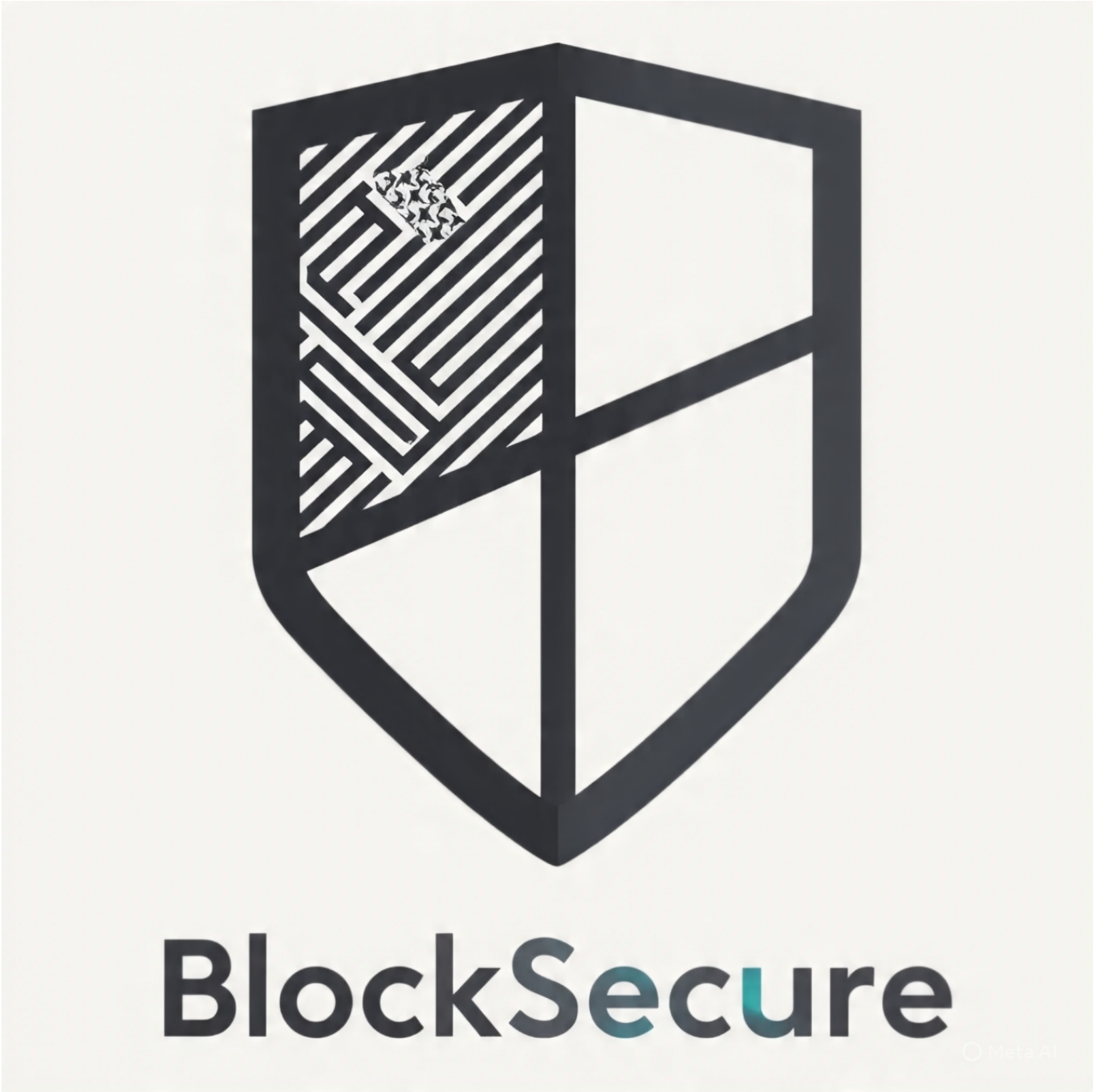

Prepared by: Ayushman

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

Ayushman 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 correspont the following commit hash:

''' 2e8f81e263b3a9d18fab4fb5c46805ffc10a9990 '''

Scope

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

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

Issues found

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

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone and is no longer private

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 private and should only be accessed through the `PasswordStore::getPassword` function, which is intended to be called only by the contract owner.

We show one such method of reading on-chain data below.

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

Proof of Concept:

The test case below demonstrates 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://187.0.0.1:8545
   ```

   You will get an output that looks like this:

   ```
   0x6d7950617373776f726400000000000000000000000000000000000000000014
   ```

   Parse that hex to a string with:

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

   You'll get:

   ```
   myPassword
   ```

Recommended Mitigation:
The contract architecture should be rethought. One approach is to encrypt the password off-chain and store the encrypted value on-chain. This would require the user to remember an off-chain decryption key. Additionally, consider removing the view function to avoid accidental transactions that expose sensitive information.

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

Description:
The `PasswordStore::setPassword` function is marked `external`, but the natspec and the contract's intended behavior indicate that only the owner should be allowed to call it.

```
function setPassword(string memory newPassword) external {
    //@audit - There are no access controls
    s_password = newPassword;
    emit SetNetPassword();
}
```

Impact:
Anyone can set or change the password, severely breaking the 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);

    vm.prank(owner);
    string memory actualPassword = passwordStore.getPassword();
    assertEq(actualPassword, expectedPassword);
}
```

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

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

Medium

Low

Informational

[I-1] The `PasswordStore::getPassword` natspec references a non-existent parameter

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 natspec for `PasswordStore::getPassword` incorrectly references a parameter `newPassword`, even though the function takes no arguments.

Impact:
The documentation is misleading and incorrect.

Recommended Mitigation:
Remove the incorrect `@param` line from the natspec.

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

End of Report