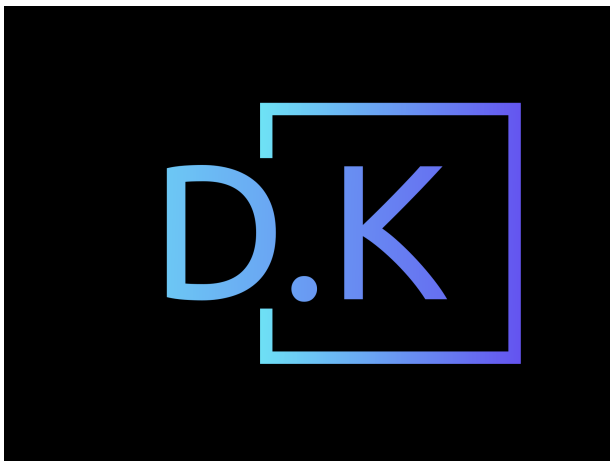

Protocol Audit Report

Version 1.0

DAVID KORGALIDZE

February 12, 2025

Protocol Audit Report

David Korgalidze

February 11, 2025

Prepared by: D.K Lead Auditors:

- David Korgalidze

Table of Contents

- Informational
  - [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.
- Likelihood and Impact:

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's passwords. The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access this password.

Disclaimer

This audit report is provided for informational purposes only. It is not a guarantee of security or functionality. The findings and recommendations are based on the current state of the code and may not cover all potential risks. Use this report at your own discretion, and ensure thorough testing and review before deploying any smart contract. The auditors are not liable for any damages or losses resulting from the use of this report or the audited code.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	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:

Scope

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

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

This audit assessed the security of the PasswordStore smart contract. Three issues were identified:

```
1.High Severity: Password stored on-chain, making it publicly visible.
2.High Severity: No access control on the setPassword function,
   allowing unauthorized password changes.
3.Informational: Incorrect natspec documentation in the getPassword
   function.
```

The recommended mitigations include implementing access controls, securing password storage off-chain, and fixing documentation inconsistencies.

Issues found

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

Severity	Number of issues found
Total	3

Findings

High

[H-1] Storing the password on-chain makes it visable to anyone,and it is not private.

Description: All data stored on chain is public and visible to anyone. The `PasswordStore::s_password` variable is intended to be hidden and only accessible by the owner through the `PasswordStore::getPassword` function.

Impact: Anyone is able to read the private password, severely breaking the functionality of the protocol.

Proof of Concept: the below test case shows how anyone could read the password from the bolockchain.

1. Create a locally running chain

```
1    make anvil
```

2. Deploy the contract to the chain.

```
1   make deploy
```

3. Run the storage tool

We use 1 because that's the storage slot of s_password in the contract.

```
1 cast storage <ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

4. You'll get an output that looks like this:

```
1 0x6d7950617373776f726400000000000000000000000000000000000000000014
```

5. You can then parse that hex to a string with:

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

6. And get an output of:

```
myPassword
```

Recommended Mitigation:

Secure Off-Chain Storage:

Avoid storing sensitive information, such as passwords, directly on-chain since all blockchain data is publicly accessible. Instead, store passwords securely off-chain using encrypted storage solutions (e.g., secure backends, encrypted databases, or secure key management systems).

On-Chain Reference to Encrypted Data:

If referencing the password is necessary on-chain, store only the cryptographic hash of the password (using SHA-256 or Keccak256) instead of the plaintext value. This approach allows verification without revealing the actual password.

Example Implementation:

Modify the contract to store the password hash and verify the user's input:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;

contract PasswordStore {
    address private s_owner;
    bytes32 private s_passwordHash;

    event PasswordUpdated(bytes32 passwordHash);

    constructor() {
        s_owner = msg.sender;
    }

    function setPassword(string memory newPassword) external onlyOwner
        {
        s_passwordHash = keccak256(abi.encodePacked(newPassword));
        emit PasswordUpdated(s_passwordHash);
    }

    function verifyPassword(string memory password) external view
       onlyOwner returns (bool) {
        return keccak256(abi.encodePacked(password)) == s_passwordHash;
   }

}
```

Likelihood and Impact:

- Impact: High
- LikeliHood: High
- 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 purpose of the smart contract and function's natspec indicate 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 SetNewPassword();
}
```

Impact: Anyone can set/change the stored password.

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

```
 function test_anyone_can_set_password(address randomAddress) public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory randomAddressPassword = "newPassword";
        passwordStore.setPassword(randomAddressPassword);

        vm.prank(owner);
        string memory owenrsPassword = "newPassword";
        passwordStore.setPassword(owenrsPassword);
        assertEq(randomAddressPassword, owenrsPassword);
    }
```

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

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

Likelihood and Impact:

- Impact: High
- LikeliHood: High

- Severity: High

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be 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()` 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 and Impact:

- Impact: None
- LikeliHood: High
- Severity: Informational/Gas/Non-crit

Informational: This is not a bug,but you should know (fix)..,