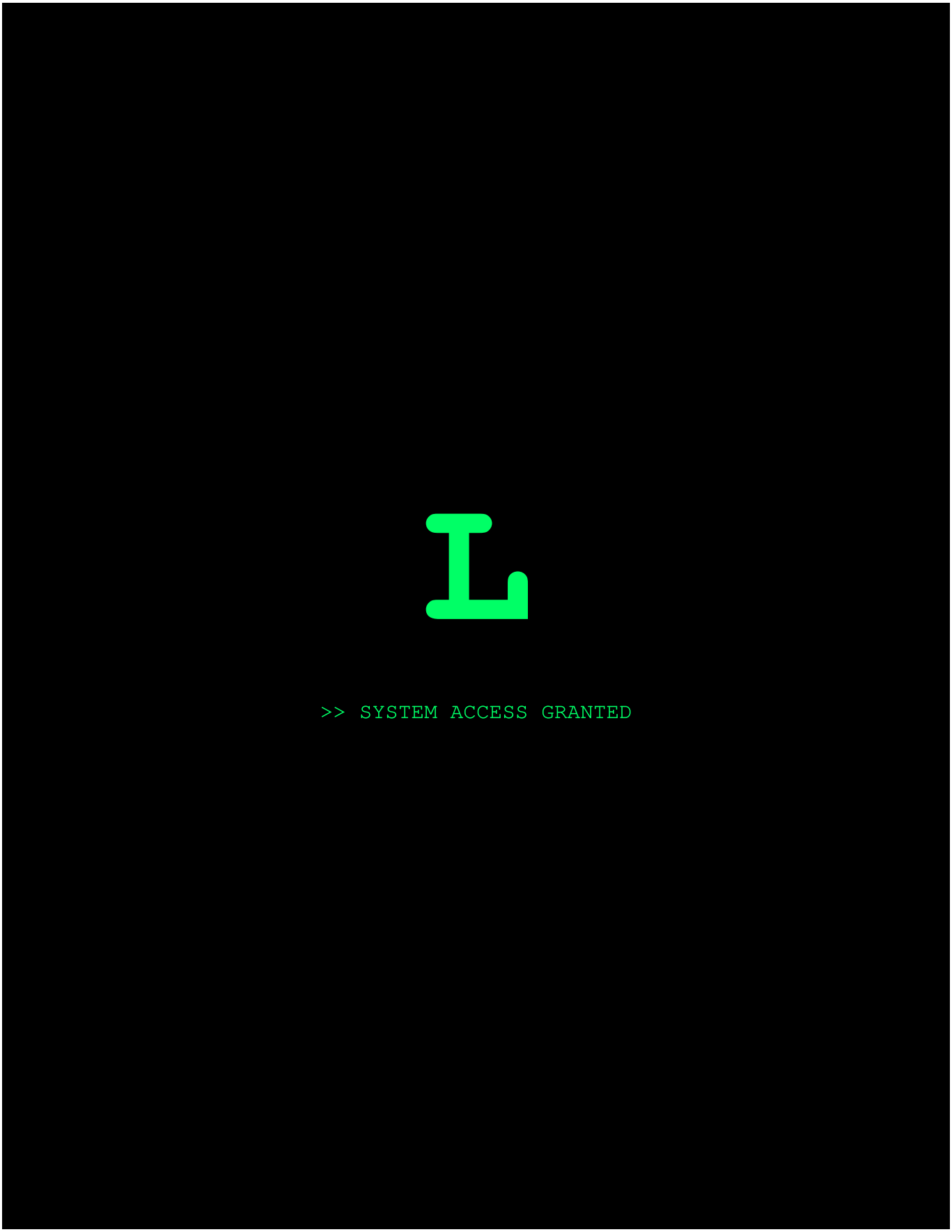

PasswordStore Audit Report

Prepared by: Lucas Silveira

Lead Security Researcher:

- Lucas Silveira

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
    - Scope
    - Roles
- Executive Summary
    - Issues found
- Findings
- High
- Low
- Informational

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

The Lucas Silveira 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

	Impact		
	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:

```
63541e54586cca26e0a69b3ddc6b8fed150e7d2d
```

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

The PasswordStore contract contains critical security flaws that compromise both confidentiality and access control. The password is stored in plaintext on-chain, making it publicly readable, and the password update function lacks access controls, allowing any user to modify it. Additionally, minor documentation issues were identified. A redesign is recommended to avoid storing sensitive data on-chain and to enforce proper authorization.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	1
Info	1
Gas Optimizations	0
Total	4

Findings

High

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

Description:
All data stored on-chain is publicly visible and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be private and accessed only through the `PasswordStore::getPassword` function, which itself is intended to be callable only by the contract owner.

However, because the password is stored directly in contract storage, anyone can read it without calling any contract functions. Below, we demonstrate one such method for reading on-chain storage directly.

Impact:
Anyone can read the private password, severely breaking the intended functionality and security guarantees of the protocol.

Proof of Concept (PoC):

The following steps show how an arbitrary user can read the password directly from the blockchain.

1. **Start a local blockchain**

```
make anvil
```

2. **Deploy the contract**

```
make deploy
```

3. **Read the contract storage**

We use slot `1` because this is the storage slot where `s_password` is stored.

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

This will return output similar to:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. **Decode the value into a string**

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

This returns:

```
myPassword
```

Recommended Mitigation:
The overall contract architecture should be reconsidered. Sensitive data such as passwords should never be stored in plaintext on-chain.

One possible approach is to encrypt the password off-chain and store only the encrypted value on-chain. This would require the user to manage an off-chain secret for decryption. Additionally, the `getPassword` view function should likely be removed to avoid accidentally exposing sensitive data via transactions.

[H-2] `PasswordStore::setPassword` lacks access control, allowing non-owners to change the password

Description:
The `PasswordStore::setPassword` function is marked as `external`. However, both the NatSpec documentation and the intended design of the contract state that **only the owner** should be able to set a new password.

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

No access control checks are performed, allowing any address to call this function.

Impact:
Any user can arbitrarily change the contract password, completely breaking the contract's intended functionality and security assumptions.

Proof of Concept:

Add the following test case 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 an access control check to ensure only the owner can call `setPassword`.

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

Low

[L-1] Initialization timeframe vulnerability

Description:
The `PasswordStore` contract exhibits an initialization timeframe vulnerability. Specifically, there is a window between contract deployment and the first call to `setPassword` during which the password remains in its default state.

Because the contract does not initialize the password in its constructor, the `s_password` variable defaults to an empty string upon deployment. During this period, the contract operates with an unset password, which may lead to unintended behavior or incorrect assumptions about the contract's state.

It is important to note that even if this issue is mitigated, the password will still remain publicly readable due to the inherent transparency of blockchain storage, as described in the **"Storing the password on-chain"** finding.

Impact:
During the initialization timeframe, the password is empty, which may expose the contract to unintended behavior or incorrect usage assumptions. While the severity is limited, this represents a deviation from expected contract initialization practices.

Recommended Mitigation:
Initialize the password during contract deployment by setting it in the constructor. The initial password value can be passed as a constructor parameter to ensure the contract starts in a valid and predictable state.

Informational

[I-1] Incorrect NatSpec for `PasswordStore::getPassword`

Description:

The NatSpec comment for `PasswordStore::getPassword` references a parameter that does not exist.

```
/*
 * @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 function signature is `getPassword()` and does not accept any parameters, making the NatSpec documentation inaccurate.

Impact:
Incorrect documentation may confuse developers, auditors, or integrators and reduce code clarity.

Recommended Mitigation:
Remove the incorrect NatSpec line.

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