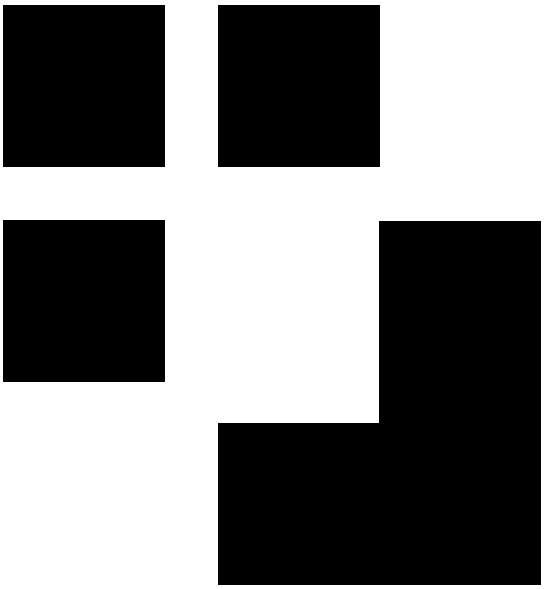

Protocol Audit Report

Version 1.0

Hexific.com

August 24, 2025

Protocol Audit Report

Hexific.com

August 24, 2025

Prepared by: Hexific Lead Security Researcher: - Febri Nirwana

Table of Contents

Protocol Summary

The PasswordStore protocol is a simple smart contract that allows a user to store a password string on-chain. The contract designates an owner at deployment, and only this owner is intended to have access to set and read the stored password.

Disclaimer

The Hexific 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:

```
0x2Ecf6aD327776bF966893C96Efb24c9747F6694B
```

Scope

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

Roles

- Owner: Who can set (`setPassword()`) and retrieve (`getPassword()`) the stored password.
- Outsiders: Any address other than the owner that should be cannot set and read the password.

Executive Summary

The Hexific team conducted a security audit of the PasswordStore smart contract. The contract is simple in scope but carries critical risks due to its flawed assumption that onlyOwner access control can provide confidentiality for sensitive data. Since all contract state is publicly visible on-chain, storing a password directly exposes it to any external observer, regardless of access modifiers.

Issues found

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

Findings

High

[H-1] Storing the password on-chain makes it visble to anyone, and no longer ptivate

Description: All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStored::s_password` variable intended to be a private variable and only accessed

through the `PasswordStored::getPassword` function, which is intended to be only called by 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 blow test case shows how anyone can read the password directly from the blockchain.

1. Create locally running chain

```
make anvil
```

2. Deploy contract to the chain

```
make deploy
```

3. Run the storage tools

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 will get an output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000`

You can the parse that hex to a string with:

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

And get the output of:

```
myPassword
```

Recommended Mitigation: Never store plaintext secrets or passwords on-chain. If password verification is required, store only the hash of the password (e.g., using `keccak256`). When a user provides their password, hash the input and stored it. This way, the actual password never exists in on-chain storage.

[H-2] Missing access control on `setPassword` allows anyone to change the password

Description:

The `PasswordStored::setPassword` function is intended to be called only by the owner to

update the password. However, the function lacks any access control modifier, which allows any external user to call it and overwrite the password value. This breaks the intended logic of the contract and removes the concept of an “owner-only” password setter.

Impact:
Any user can arbitrarily change the stored password. This undermines the protocol’s security assumptions and renders the password mechanism meaningless, as no single trusted party controls it anymore.

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

Code

```
function testExploit_SetPassword() public {
    vm.prank(attacker);
    passwordStored.setPassword("attackerPassword");

    // Attacker has successfully overwritten the password
    assertEq(passwordStored.getPassword(), "attackerPassword");
}
```

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

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn’t exist, causing the natspec tp be corrected.

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` functioon signature is `getPassword()` which the natspec say is should be `getPassword(string)`

Impact: The natspec is incorect.

Recommended Mitigation: Remove the incorect natspec line.

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