Protocol Audit Report

Zntb

tibeth

Jun 25, 2025

tibeth

Protocol Audit Report

Version 1.0

zntb.io

June 25, 2025

Protocol Audit Report

Zntb

Jun 25, 2025

Prepared by: Zntb Lead Security Resea- Table of Contents

Table of Contents

Protocol Summary

PasswordStore is a protocol 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 zntb 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

Commit Hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
./src/└──
 PasswordStore.sol
```

Roles

- Owner: The user who can set the password and read 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 on-chain makes it visible to anyone, and is 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 should only be accessible through the `PasswordStore::getPassword` function.

Below we demonstrate one method of reading this data from the blockchain.

Impact: Anyone can read the private password, severely compromising the protocol's functionality.

Proof of Concept:

The following test case demonstrates how anyone can read the password directly from the blockchain:

1. Create a locally running chain:

```
make anvil
```

2. Deploy the contract:

```
make deploy
```

3. Run the storage tool (using slot 1 as it corresponds to `s_password` in the `PasswordStore` contract):

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

This will output something like:

```
0x6d79506173737776f7264000000000000000000000000000000000000000014
```

4. Parse the hex output to a string:

```
cast parse-bytes32-string 0
    x6d79506173737776f7264000000000000000000000000000000000000000014
```

Which will return:

```
myPassword
```

Recommended Mitigation: The contract architecture should be reconsidered. One approach would be to encrypt the password off-chain and store only the encrypted version on-chain. This would require users to remember a separate decryption key. Additionally, consider removing the view function to prevent accidental exposure of the decryption key in transaction data.

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

Description: The `PasswordStore::setPassword` function is marked as `external` without any access controls, despite the NatSpec comment stating: "This function allows only the owner to set a new password."

```
function setPassword(string memory newPassword) external {
    // @Audit - No access controls implemented
    s_password = newPassword;
    emit SetNewPassword();
}
```

Impact: Any user can set or change the stored password, completely undermining the contract's intended functionality.

Proof of Concept:

Add the following test to `PasswordStore.t.sol`:

Code

```
function test_anyone_can_set_password(address randomAddress) public {
    vm.assume(randomAddress != owner);
    vm.startPrank(randomAddress);
    string memory expectedPassword = "myNewPassword";
    passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation: Add access control checks to the function:

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

Informational

[I-1] Incorrect NatSpec in `PasswordStore::getPassword` references non-existent parameter

Description: The NatSpec documentation for `PasswordStore::getPassword` incorrectly references a `newPassword` parameter that doesn't exist in the function signature.

```
/*
 * @notice This allows only the owner to retrieve the password.
 * @param newPassword The new password to set.
 */
function getPassword() external view returns (string memory) {}
```

Impact: The documentation is misleading and incorrect.

Recommended Mitigation: Remove the incorrect parameter documentation:

```
/*
 * @notice This allows only the owner to retrieve the password.
- * @param newPassword The new password to set.
 */
```
