



Protocol Audit Report

Version 1.0

azduR

January 17, 2025

Protocol Audit Report

azduR

January 17, 2025

Prepared by: azduR Lead Auditors: - xxxxxxxx

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
 - Scope
 - Roles
- Executive Summary
 - Issues found
- Findings
 - High
 - * [H-1] Storing the password on-chain makes it visible to anyone and no longer private
 - * [H-2] `PasswordStore::setPassword` function has no access controls, meaning a non-owner could change the password
 - Informational
 - * [I-1] The `PasswordStore::getPassword` function natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect
 - Gas

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 isn't designed to be used by multiple users. Only the owner should be able to set and access this password.

Disclaimer

The YOUR_NAME_HERE 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
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

Scope

Roles

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

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

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

Proof of Concept: The below test case shows how anyone can read the password directly from the blockchain. We used Foundry's cast tool to read directly from the storage of the contract, without being the owner

- 1) Create a locally running chain `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://127.0.0.1:8545`
- 4) You'll get an output that looks like this: `0x6d7950617373776f726400`
- 5) You can then parse that hex to a string with: `cast parse-bytes32-string 0x6d7950617373776f72640014`
- 6) And get an output of: `myPassword`

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain, and then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

[H-2] `PasswordStore::setPassword` function 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.`

Impact: Anyone can set/change the password of the contract, severely breaking the contract intended functionality

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

```
1 function test_anyone_can_set_password(address randomAddress) public {
2     vm.assume(randomAddress != owner);
3     vm.prank(randomAddress);
4     string memory expectedPassword = 'myNewPassword';
5     passwordStore.setPassword(expectedPassword);
6
7     vm.prank(owner);
8     string memory actualPassword = passwordStore.getPassword();
9     assertEq(actualPassword, expectedPassword);
10 }
```

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

```
1 if (msg.sender != s_owner){
2     revert PasswordStore_NotOwner();
3 }
```

Informational

[I-1] The PasswordStore::getPassword function natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Description:

```
1      /*
2      * @notice This allows only the owner to retrieve the password.
3  @>   * @param newPassword The new password to set.
4      */
5      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.

Proof of Concept:

Recommended Mitigation: Remove the incorrect natspec line.

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

Gas