Protocol Audit Report

Version 1.0

Chornyi.io

July 21, 2024

Protocol Audit Report

Artem Chornyi

July 21, 2024

Prepared by: Artem Chornyi Lead Auditors: - Artem Chornyi

Table of Contents

Protocol Summary

PasswordStore is a protocol designed to store and recover user passwords. The protocol is intended for use by a single user and is not intended for use by multiple users. Only the owner should be able to set and access this password.

Disclaimer

The Artem Chornyi 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

The findings described in this document correspond the following 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

The security audit for the PasswordStore smart contract was conducted to evaluate its security posture and identify potential vulnerabilities. This audit was performed by a dedicated team of one security researcher over the course of two day. The team conducted a thorough review, including manual code analysis and automated security testing tools, to ensure comprehensive coverage of potential security issues.

Issues found

Severity	Number of issues dound
High	2
Medium	0
Low	1
Info	1
Total	4

Findings

High

[H-1] Passwords Stored On-Chain are Visible to Everyone

Description: All data stored on the blockchain is publicly accessible and can be directly read by anyone. The `PasswordStore::s_password` variable is designed to be private and should only be accessed

via the `PasswordStore::getPassword` function, which is intended for use only by the contract owner. However, this variable can be read directly through various off-chain methods.

Impact: The password is not private.

Proof of Concept: The following test case demonstrates how anyone can read the password directly from the blockchain. We use the foundry's cast tool to read from the contract's storage without being the owner.

1. Create a locally running chain: `bash make anvil`
2. Deploy the contract to the chain: `bash make deploy`
3. Run the storage tool: `bash cast storage <ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545`

This command returns an output similar to: `0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can convert this hex value to a string with: `bash cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000014`

Resulting in: `myPassword`

Recommended Mitigation: The contract's overall architecture should be revised. One approach is to encrypt the password off-chain and store only the encrypted version on-chain. This would require users to remember an off-chain password to decrypt the stored password. Additionally, removing the view function is advisable to prevent users from accidentally revealing the password in a transaction.

[H-2] `PasswordStore::setPassword` is Callable by Anyone

Description: The `PasswordStore::setPassword` function is declared as `external`, but the function's documentation states that only the owner should be able to set a new password.

```
1     function setPassword(string memory newPassword) external {
2         // @audit - There are no access controls here
3         s_password = newPassword;
4         emit SetNetPassword();
5     }
```

Impact: Anyone can set or change the contract's password.

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

```
1 function test_anyone_can_set_password(address randomAddress) public {
2     vm.prank(randomAddress);
3     string memory expectedPassword = "myNewPassword";
```

```
4       passwordStore.setPassword(expectedPassword);
5       vm.prank(owner);
6       string memory actualPassword = passwordStore.getPassword();
7       assertEq(actualPassword, expectedPassword);
8   }
```

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

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

Medium

Low

[L-2] Initialization Timeframe Vulnerability

Description: The PasswordStore contract has a vulnerability during the initialization timeframe. Between contract deployment and the explicit call to `setPassword`, the password remains in its default state. Even after fixing this issue, the public nature of blockchain data means the password's visibility cannot be completely hidden.

Impact: During the initialization timeframe, the contract's password is empty, potentially allowing unauthorized access or unintended behavior.

Recommended Mitigation: To address the initialization timeframe vulnerability, set an initial password value during the contract's deployment (in the constructor). This value can be passed as a parameter to the constructor.

Informational

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

Description: The natspec comment for the `PasswordStore::getPassword` function incorrectly indicates a parameter that does not exist, causing the documentation to be misleading.

```
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) {
```

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line:

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

Gas