Protocol Audit Report

Version 1.0

Parth

February 26, 2024

Protocol Audit Report

Parth Sharma

january 20, 2024

Prepared by: Parth

Lead Security Researcher:

- Parth Sharma

Table of Contents

Protocol Summary

PasswordStore protocol is designed to store and retrieve the password of user/owner. Only owner of the contract can set or access the password.

Disclaimer

We 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 the document correspond to the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

./src/

#– PasswordStore.sol

Roles

Owner: The user who can set the password and read the password.

Issues found

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

Findings

High

[H-1] TITLE Password stored on chain visible to anyone, it's not private.

Description: All data stored on-chain is visible to anyone, and can be read directly from blockchain. The `passwordStore::s_password` variable is intended to be a private an do only accessed through the `passwordStore::getPassword` function, which is intended to be called by only owner of the contract.

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

Impact: Password is read by anyone, severly breaking the functionality of the protocol.

Proof of Concept: (Proof of code) we need a local chain running.

```
forge anvil
```

Now we have to deploy our contract on-chain.

```
forge script script/DeployPasswordStore.s.sol:DeployPasswordStore --rpc
   -url http://127.0.0.1:8545 --private-key 0
   xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80
```

Foundry allows us to check the storage of a deployed contract with a very simple cast command. Storage slot of `PasswordStore::s_password` is 1.

we run the command `cast storage <address of contract> <storage slot>`:

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1
```

Following is the output in bytes form: `0x6d7950617373776f7264000000000000000000000000000000000`

By using another convenient Foundry command we can now decode this data:

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

Our output then becomes: `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.

[H-2] TITLE Non-Owner can set the password, anyone can manipulate the `PasswordStore::s_password` variable.

Informational

[I-1] TITLE The `passwordStore::getPassword()` natspec indicates the parameter that doesn't exist, causing the natspec to be incorrect

Description:

```
    /*
     * @notice This allows only the owner to retrieve the password.

     @audit newPassword is not the parameter in the following function.

@>     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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