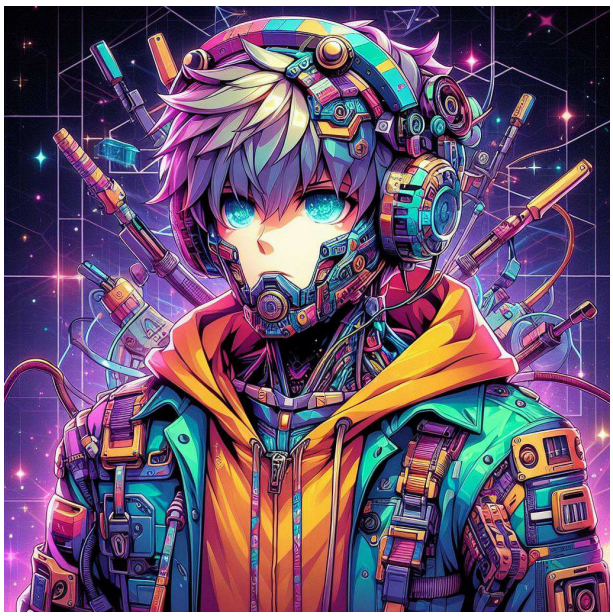

Protocol Audit Report

Version 1.0

0xmazode

October 31, 2024

Protocol Audit Report

Mazode

October 31, 2024

Prepared by: Mazode Lead Auditors: - Mazode

Table of Contents

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 is not designed to be used by multiple users. Only the owner should be able to set and access this password.

Disclaimer

The Mazode 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

Audit Details

The findings described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract.

Executive Summary

Issues found

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

Findings

High

[H-1] Password is visible to everyone and should not be stored on-chain

Description: The data stored on-chain is visible to everyone and can be read from the blockchain. The `PasswordStore::s_password` variable is supposed to be a private variable and should only be accessed through `PasswordStore::getPassword()` function which only the owner of the contract can call.

We have demonstrated a similar method of reading any data off chain below.

Impact: Anyone can read the private password, which could severely affect the protocol's functionality.

Proof of Concept: (Proof of Code)

The following test case showcases how anyone can read the private password directly from the blockchain.

1. Create a locally running chain

```
make 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 PasswordStore contract.

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

You'll get output similar to this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string like this:

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

Which will result in a output like this:

```
myPassword
```

Recommended Mitigation: Since the password is stored in a way that allows anyone to read it, the contract's security is compromised. To mitigate this risk, consider using Zero-Knowledge Proofs (ZKPs). ZKPs are powerful cryptographic methods that enable one party (the "prover") to demonstrate to another party (the "verifier") that they know a specific piece of information without revealing the information itself. By implementing ZKPs, sensitive data like passwords can be verified without exposing them on-chain, significantly enhancing the security and privacy of the contract.

[H-2] `PasswordStore::setPassword` has no access control resulting in password change by a non-owner

Description: The `PasswordStore::setPassword` function does not have any access control meaning that a non-owner user could potentialy set a new password. The function is supposed to be only called by the owner.

```
1       function setPassword(string memory newPassword) external {
2   >@      // @audit Missing access control
3           s_password = newPassword;
4           emit SetNetPassword();
5       }
```

Impact: Anyone can alter the contract's functionality by changing the password of the contract.

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

Code

```
1       function test_non_owner_can_set_password(address randomAddress)
            public {
2           vm.assume(randomAddress != owner);
3           vm.prank(randomAddress);
4           string memory expectedPassword = "pass12";
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 conditional to the `setPassword` function.

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

Medium

Low

Informational

[I-1] The `PasswordStore::getPassword` 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()` which the NatSpec say should be `getPassword(string)`.

Impact: The NatSpec is incorrect

Recommended Mitigation: Remove the incorrect NatSpec line

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

Gas