Password store Audit Report

DecentralizedGlasses(Sivaji)

Jan 12, 2026

Protocol Audit Report

Version 1.0

Cyfrin.io

January 12, 2026

Password store Audit Report

DecentralizedGlasses(Sivaji)

Jan 12, 2026

Prepared by: [DecentralizedGlasses] Lead Auditors: - Sivaji

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
- High -
  - Likelihood & Impact:
    - [H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password
  - Likelihood & Impact:
- Medium
- Low
- Informational
  - [L-1] TITLE `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect
  - Likelihood & Impact:
- Gas

Protocol Summary

Protocol does stores the passwords of the user set by the user itself.

Disclaimer

The DedentralizedGlasses(Sivaji) 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

This is the security review of the password store porotocl, in the whole journey of finding vulnerabilities. I found 2 High and one Informational severity.

Scope

```
src:
|__ PasswordStore.sol
```

Roles

- Owner: the owner role is for setting the password
- user: The user is also owner who fethces the password

Executive Summary

- The core protocol usage is to store the password of the user set by the user itself, and when fetching the password it should be able to fetch only by the owner who set the password.

Issues found

I found 2 high severity and 1 Informational vulnerabilities, The user is able to set the password but the issue is, it is fetched by anyone on-chain. The another high is any user can change the any user password wihtout the owner consent, and also there is little natspec error that says that it need to password as parameter, but it isn't passing any parameter for that function.

Findings

High

1.

[H-1] Stroing the password on-chain makes it visible to everyone, and no longer priavte

Description: The all variables stored on-chain are visible to anyone, can read directly in blockchain. The `PasswordStore::s_passwordariable` is intedned to be a private varaible nd only accessed by `PasswordStore::getPassword()` function and intended to be only called by owner

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

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

Proof of Concept: (Proof of code) The below test proves that this protocol is failing in this issue

1. First we need a local chain running.

`anvil` Note: Most PoC's won't require a local blockchain

2. Next we need to deploy our protocol, fortunately, PasswordStore has a make command set up for us. Note that their deploy script is setting the password myPassword in the process. Open a new terminal and run the following.

`make deploy`

- Foundry allows us to check the storage of a deployed contract with a very simple cast command.
- For this we'll need to recall to which storage slot the s_password variable is assigned.

proof-of-code1

- With this consideration we can run the command cast storage like this (your address may be different).

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1
```

- We should receive an output similar to this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

- This is the bytes form of the data at storage slot 1. By using another convenient Foundry command we can now decode this data.

```
cast parse-bytes32-string 0x6d7950617373776f72640000000000000000000000000000000000000000001
```

- Our output then becomes:

```
myPassword
```

- And we've done it. In a few quick commands we've shown that the data our client is expecting to keep hidden on chain is accessible to anyone. Let's add these steps as proof to our report. Things are getting long, so I've collapsed the report examples going forward!

3. Finding Report

- Wrap Up We've one more section in our report to fill out, the Recommended Mitigations. This is where we get a chance to illustrate our experience and bring value to the process by offering our expert advice on how rectify the problems faced by this vulnerability.

Recommended Mitigation: Due to this issue overall protocol will get an big threat, the threat is all about when a user can run a chain locally and and can decrypt the password. From the contract address it will give you the variables that are stored in specific slots and our password is in slot number 1, you will get a hash after and running the script and that hash is used to cast the real passowrd. However you're also likely want to remove e view function as you wouldn't want the user send the transaction with this decryption key.

Likelihood & Impact:

Impact : HIGH Likelihood: HIGH Severity: HIGH

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password

Description: The `PasswordStore::getPassword` is set external function. This is a big threat to all the users. However the natspec of this function allows `only owner` to set the `new password`.

Impact: Anyone can set/change the password, which is severely breaking the contract's intended functionality.

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

Code

```
    function test_anyone_can_set_password(address nonOwner) public {
        vm.assume(nonOwner != owner);
        vm.startPrank(nonOwner);
        string memory password = "HackedPassword";
        passwordStore.setPassword(password);
        vm.stopPrank();

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

Recommended Mitigation: Add an access conrol to the conditional to the `setPassword` function.

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

Likelihood & Impact:

Impact : HIGH Likelihood: HIGH Severity: HIGH

Medium

None # Low None # Informational ### [L-1] TITLE `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Description:

```
  /*
     * @notice This allows only the owner to retrieve the password.
>>     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

The `PasswordStore::getPassword()` function signature is `getPassword()` while the natspec says it should be `getPAssword(string)`.

Impact: The natspec the incorrect

Proof of Concept:

Recommended Mitigation: Remove the incorrect natpsec

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

Likelihood & Impact:

Impact : None Likelihood: HIGH Severity: Informational/Gas/Non-crits # Gas