



Protocol Audit Report

Version 1.0

Gush

February 18, 2025

Protocol Audit Report

Gush

Feb 18, 2025

Prepared by: Gush Lead Security Researcher: - Gush

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` has no access controls, meaning a non-owner could set the password
 - Informational
 - * [I-1] The NatSpec documentation for `PasswordStore::getPassword` indicates a non-existent parameter, causing the documentation to be incorrect.

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 Gush's 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

```
1 7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
1 ./src/  
2 #-- 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

Add some notes about how the audit went, types of things you found, etc...

We spent X hours with Z auditors using Y tools, etc...

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 called only 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 following test case shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain: `bash make anvil`
2. Deploy the contract to the chain: `bash make deploy`
3. Run the storage tool:

We use 1 because that's the storage slot of `PasswordStore::s_password` in the contract.

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

You will get an output similar to this:

[illegible][illegible]

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 password. However, you would also likely want to remove the view function, as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

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

Description: The `PasswordStore : : setPassword` function is set to be an `external` function. However, the `NatSpec` of the function and the overall purpose of the smart contract indicate that “This function allows only the owner to set a new password.”

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

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

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

Code

```
1 // @audit The function setPassword is not restricted to the owner,
  allowing anyone to set the password.
2 function test_anyone_can_set_password(address randomAddress) public {
3     vm.startPrank(randomAddress);
4     string memory expectedPassword = "myNewPassword";
5     passwordStore.setPassword(expectedPassword);
6
7     vm.startPrank(owner);
8     string memory actualPassword = passwordStore.getPassword();
9     assertEq(actualPassword, expectedPassword);
10 }
```

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

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

Informational

[I-1] The NatSpec documentation for `PasswordStore::getPassword` indicates a non-existent parameter, causing the documentation 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 function signature is `PasswordStore::getPassword`, while the NatSpec documentation indicates it should be `PasswordStore::getPassword(string)`.

Impact: The NatSpec documentation is incorrect.

Recommended Mitigation: Remove the incorrect NatSpec documentation line.

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