PasswordStore Protocol Audit Report

Version 1.0

SymmaTe

February 11, 2026

Protocol Audit Report

SymmaTe

February 11, 2026

Prepared by: SymmaTe(https://github.com/SymmaTe) Lead Security Researcher: - SymmaTe

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 owner, who can set and retrieve their password. Only the owner should be able to set and retrieve the password.

Disclaimer

The SymmaTe 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
	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 this document correspond to the following commit hash:

```
7d55682ddc4f9b748262c638e4d550b4a8c4b032
```

Scope

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

Roles

- Owner: The user who can set and retrieve the password.
- Outsiders: No one else should be able to set or retrieve the password.

Executive Summary

Category	Details
Solidity Version	0.8.18
Chain(s)	Ethereum
nSLOC	20

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Informational	1
Gas	0
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 directly read 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 only called 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: (Proof of Code)

The below test case shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract on chain

```
make deploy
```

3. Run the storage tool

We use 1 because that is the storage slot of `s_password` in the contract.

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

You will get the output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse the hex to a string with:

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

And get an output of:

```
myPassword
```

Recommended Mitigation: Due to the nature of the blockchain, all data stored on-chain is publicly accessible and cannot be truly hidden. The overall architecture of the `PasswordStore` protocol should be rethought. One approach would be to encrypt the password off-chain before storing it on-chain, so that only users with the decryption key can retrieve the original password. However, it should be noted that this would also require a mechanism for secure key management off-chain.

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change 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 that only the owner can set a new password.`

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

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

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

Code

```
1      function test_anyone_can_set_password(address randomAddress) public
           {
2          vm.assume(randomAddress != owner);
3          vm.startPrank(randomAddress);
4          string memory expectedPassword = "myNewPassword";
5          passwordStore.setPassword(expectedPassword);
6          vm.stopPrank();
7
8          vm.startPrank(owner);
9          string memory actualPassword = passwordStore.getPassword();
10         vm.stopPrank();
11         assertEq(actualPassword, expectedPassword);
12     }
```

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

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

Medium

None.

Low

None.

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()` while the natspec suggests it 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

None.