PasswordStore Initial Audit Report

Version 0.1

author: 0xZaHeD

September 14, 2025

PasswordStore Audit Report

FOXWAR

September 14, 2025

PasswordStore Audit Report

Prepared by:

- 0xZaHeD

Assisting Auditors:

- None

Table of contents

See table

- Findings
  - High
    * [H-1] Sensitive Data Exposed On-Chain (Storing plaintext password makes it publicly readable)
    * [H-2] Missing Access Control in `PasswordStore::setPassword` (Anyone can change stored password)
  - Info
    * [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Disclaimer

The FOXWAR team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the 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

Audit Details

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

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

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.

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] Sensitive Data Exposed On-Chain (Storing plaintext password makes it publicly readable)

Description: The variable `PasswordStore::s_password` is declared **`private`** but still stored on-chain in plaintext. In Ethereum, all contract storage is publicly accessible, so anyone can read this value directly.

Impact: Attackers can easily extract the password from storage, bypassing intended secrecy.

Proof of Concept:

1. Deploy the contract locally with `s_password = "myPassword"`.

```
1   make deploy
```

→ get contract address

2. Run:

```
1 cast storage <contract address> 1 <--rpc-url>
```

→ returns <0x6d7950617373776f726400000000000000000000000000000000000000000014>

3. Decode:

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

→ reveals `myPassword`.

Recommended Mitigation: - Do not store sensitive information like passwords on-chain.

- If validation is required, store a hash (e.g., `keccak256(password)`) and compare hashes instead of the actual value.
- Use off-chain secret management solutions when true secrecy is required.

[H-2] Missing Access Control in `PasswordStore::setPassword` (Anyone can change stored password)

Description: The function `PasswordStore::setPassword` is documented to allow only the owner to set a new password, but it lacks any ownership check. As a result, any external address can call this function and overwrite the stored password.

```
    function setPassword(string memory newPassword) external {
@>      // @audit - Missing access control here
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: An attacker/anyone can call `PasswordStore::setPassword` and replace the password with arbitrary values.

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

Code

```
function test_nonOwner_can_set_password() public {
    vm.assume(nonOwner != owner);
    vm.prank(nonOwner);
    string memory expectedPassword = "myNewPassword";
    passwordStore.setPassword(expectedPassword);

    vm.prank(owner);
    string memory actualPassword = passwordStore.getPassword();

    assertEq(actualPassword, expectedPassword);
}
```

Recommended Mitigation: - Restrict access to `PasswordStore::setPassword` by adding an ownership check.
- Use a modifier like `onlyOwner` to enforce access control.

```
modifier onlyOwner() {
            require(msg.sender == owner, "Not authorized");
            _;
        }
```

- Ensure the contract's `owner` is set correctly in the constructor.
- If more roles are needed, use a role-based access control system (e.g., OpenZeppelin `AccessControl`).

Info

[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 natspec for the function `PasswordStore::getPassword` indicates it should have a parameter with the signature `getPassword(string)`. However, the actual function signature is `getPassword()`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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