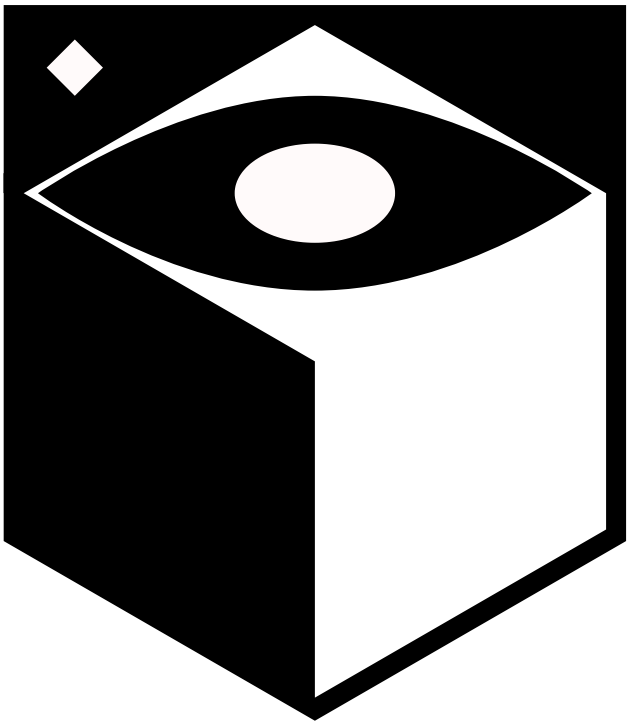

Protocol Audit Report

Prepared by: Ultrakelevra

2025-10-22

Table of Contents

Protocol Summary

The PasswordStore protocol allows a user to set and retrieve a password. The password is stored on-chain and can only be set and retrieved by the owner.

Disclaimer

The Ultrakelevra 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

- Commit Hash: `be47a4de6d0ccb6621e075caee60409c3f7eac28`
- Repo URL: `https://github.com/Cyfrin/3-passwordstore-audit`
- Solc Version: **0.8.18**
- Chain(s) to deploy contract to: **Ethereum**

Scope

```
./src/
└── 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

This audit is part of the Cyfrin Smart Contract Security Review course. I spent 2 days on this audit (I could have finished the subject in a single day but ran out of time).

Issues found

Severity	Number of findings
High	2
Medium	0
Low	0
Informational	2
Gas	0
Total	4

Findings

High

[H-1] Private storage variable is publicly readable on the blockchain

Description: The `s_password` variable in `PasswordStore.sol` is declared as `private`, but this only provides namespace privacy within Solidity. On the blockchain, storage is publicly readable by anyone. The

`private` keyword does not provide any actual security or encryption. It merely makes the variable inaccessible to other smart contracts.

Impact: This is a critical vulnerability that completely invalidates the protocol's primary security mechanism. Anyone can read the private password directly from the contract storage, making the access control in `getPassword()` meaningless. The password is stored in plain text on-chain and is permanently visible to all blockchain participants.

Proof of Concept: To prove that anyone can read the variable, we can follow the steps below:

1. Deploy the contract and set a password:

```
make anvil
make deploy
```

2. Read the password from storage slot 1 (where `s_password` is stored):

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

Output (example):

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

3. Decode from **bytes32** to **string**:

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

Output:

```
myPassword
```

Recommended Mitigation: If on-chain storage is absolutely required:

- Encrypt the password before storage.
- Require users to manage encryption keys off-chain.
- Consider generating the password off-chain in the user's client application with the off-chain + on-chain fragments. That way, the off-chain password never even leaves the user's device.

[H-2] Missing access control allows anyone to set the password

Description: The `setPassword()` function in `PasswordStore.sol` lacks access control, allowing any address to set the password. This completely undermines the contract's intended functionality where only the owner should be able to modify the password.

Impact: This is a critical vulnerability that allows unauthorized modification of the password:

```
    function setPassword(string memory newPassword) external {
@>      // @audit - NO ACCESS CONTROLS!
        s_password = newPassword;
        emit SetNewPassword();
    }
```

Proof of Concept: Add this test to the project's test file:

▶ Test Code

```
    function anyone_can_set_password(address randomAddress) public {
        vm.assume(owner != randomAddress);
        vm.prank(randomAddress);

        string memory expectedPassword = "Hacked Password";
        passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation:

Add access control checks using the existing owner variable:

```
function setPassword(string memory newPassword) external {
+    if (msg.sender != s_owner) {
+        revert PasswordStore__NotOwner();
+    }
    s_password = newPassword;
    emit SetNewPassword();
}
```

Alternative (Recommended): Use OpenZeppelin's `Ownable` contract for more robust and standardized access control:

```
// File: PasswordStore.sol
+import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
```

```
-contract PasswordStore {
+contract PasswordStore is Ownable {
-   address private s_owner;

-   constructor() {
-       s_owner = msg.sender;
+   constructor() Ownable(msg.sender) {
    }

-   function setPassword(string memory newPassword) external {
+   function setPassword(string memory newPassword) external onlyOwner {
        s_password = newPassword;
        emit SetNewPassword();
    }
}
```

Informational

[I-1] Incorrect NatSpec documentation in `getPassword()` function

Description: The `getPassword()` function has incorrect NatSpec documentation. The documentation mentions a parameter `newPassword` that does not exist in the function signature.

Impact: The misleading documentation makes it difficult for developers and auditors to understand the function's actual behavior. This can lead to confusion during code review and integration.

Proof of Concept: `...`

Recommended Mitigation: Remove the incorrect parameter documentation from the NatSpec.

```
  /*
   * @notice This allows only the owner to retrieve the password.
-  * @param newPassword The new password to set.
   */
  function getPassword() external view returns (string memory) {
```

- **[I-2] Event lacks useful parameters for off-chain monitoring**

Description: The `SetNewPassword` event in `PasswordStore.sol` emits no parameters, making it difficult to track who set the password or when it was changed without parsing transaction data.

Impact: The tracking of users and passwords would be difficult without event information. It is possible to monitor the user's activity by keeping track of transactions, wallets and contracts, yet it would consume valuable node API credits. Most valuable data should be obtainable through a call to `eth_getLogs` in a block range and not require additional calls to the contract or ask for block information. This way, the data can be obtained without consuming additional resources.

Recommended Mitigation: Add useful parameters to the event for better observability:

```
- event SetNewPassword();
+ event SetNewPassword(address indexed user, string encryptedPassword);
```

Note: While including the password hash or other data in events might seem useful, be mindful of privacy concerns, especially given the High severity findings regarding on-chain storage of sensitive data.