PasswordStore.sol — Protocol Security Audit

Version 1.0

Albert Kacirek

October 23, 2025

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
    * [H-1] Storing password on-chain makes it publicly visible, so it is no longer private
      · Likelihood & Impact:
    * [H-2] `PasswordStore::setPassword` allows anyone to change the password
      · Likelihood & Impact:
  - Informational
    * [I-1] The NatSpec of `PasswordStore::getPassword` mentions a parameter that doesn't exist in the function
      · Likelihood & Impact:
- Conclusion

Protocol Summary

The PasswordStore protocol is a simple Solidity contract designed to allow a single owner to store, update, and retrieve a password string. It aims to restrict access to the password to the owner only, using a private storage variable and basic getter/setter functions.

Disclaimer

I have made every effort to find as many vulnerabilities in the code in the given time period, but assume no liability for the findings provided in this document. This security audit 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 in this document correspond to this commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

Roles

Owner: The user who can set the password and read the password. Others: No one else should be able to set or read the password.

Executive Summary

The `PasswordStore` contract is intended to allow an owner to securely store and update a password. However, the audit found that the current design does not achieve this goal. Two high-severity issues were identified: - Passwords stored on-chain are publicly visible. - The `setPassword` function lacks access control, allowing anyone to modify the stored value.

As a result, the contract fails to provide confidentiality or integrity guarantees. It is not recommended for production use without a complete redesign to remove on-chain secret storage.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing password on-chain makes it publicly visible, so it is no longer private

Description:
All storage variables are publicly visible on-chain, so even when declared as private, the `PasswordStore::s_password` variable is not truly private and can be read by anyone, not just through the `PasswordStore::getPassword` function, but directly by reading the contract's storage. An example is shown in the Proof of Concept section below.

Impact:
Anyone can read the value of `s_password`, breaking the security of the protocol. This undermines the purpose of the contract.

Proof of Concept:
You can verify that anyone can read the stored password with this process:

1. Run a local testnet (such as Anvil via Foundry):

   ```
   make anvil
   ```

2. Deploy the contract:

   ```
   make deploy
   ```

3. Use `cast storage` to view the contents of `s_password` (which is found at the storage slot 1):

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

4. Convert the output in hex to string:

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

Note:
This example works for short strings (<= 31 bytes). Longer strings are stored beginning at keccak256(abi.encode(uint256(1))) and can still be retrieved using additional cast storage calls for subsequent slots.

Recommended Mitigation:
It is not possible to store private information securely on-chain. To handle passwords safely:

1. Never store or transmit plaintext secrets on-chain.
2. If password verification is required, store only a hash (e.g., keccak256(password)), and verify inputs against it.
3. Alternatively, encrypt and store the password off-chain (e.g., in an external database or IPFS) using a key that never touches the blockchain.
4. Consider removing the getPassword function entirely to prevent accidental on-chain disclosure.

Likelihood & Impact:

- Likelihood: HIGH
- Impact: HIGH
- Severity: HIGH

[H-2] `PasswordStore::setPassword` allows anyone to change the password

Description:
The `PasswordStore::setPassword` function does not restrict access to the owner, allowing anyone to change the password. This contradicts the NatSpec line:
`@notice This function allows only the owner to set a new password.`

```
function setPassword(string memory newPassword) external {
    // @audit - Missing access control
    s_password = newPassword;
    emit SetNetPassword(); // typo (should be SetNewPassword)
}
```

Impact:
Anyone can change the `s_password` variable through the `setPassword` function, breaking the contract's intended functionality.

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

```
function test_anyone_can_set_password() public {
    vm.assume(owner != address(1));
    vm.prank(address(1));
    string memory expectedPassword = "hackedPassword";
    passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation:
Since storing any private data on-chain in plain text is not secure, this function is not recommended. If you intend to keep this function, add an address control check to the `setPassword` function, using either `revert` or `require`. Using `revert` is more gas-efficient in this case.

```
function setPassword(string calldata newPassword) external {
    if (msg.sender != s_owner) revert PasswordStore__NotOwner();
    s_password = newPassword;
    emit SetNewPassword(); // event originally named "SetNetPassword" -
        likely a typo
}
```

Note:
This fixes integrity only. Storing plaintext on-chain is still publicly readable from storage; if secrecy matters, do not store plaintext and consider storing only a salted hash or moving secrets off-chain.

Likelihood & Impact:

- Likelihood: HIGH
- Impact: HIGH
- Severity: HIGH

Informational

[I-1] The NatSpec of `PasswordStore::getPassword` mentions a parameter that doesn't exist in the function

Description:
There is no newPassword to be set in `PasswordStore::getPassword`. The function signature is `getPassword()`, while NatSpec suggests it should be `getPassword(string)`.

Impact:
The NatSpec is incorrect. The function doesn't allow changing a password, it only allows viewing it.

Recommended Mitigation:
Remove the incorrect NatSpec line:

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

Likelihood & Impact:

- Likelihood: HIGH
- Impact: NONE
- Severity: Informational/Gas/Non-critical

Conclusion

The `PasswordStore` contract contains critical design flaws that compromise confidentiality and integrity. Passwords stored on-chain cannot be private, and missing access control allows arbitrary modification. It is recommended to avoid storing any sensitive information on-chain and to redesign the contract around secure, off-chain password management.