Protocol Audit Report

Version 1.0

xByteNeo

August 7, 2025

Protocol Audit Report

xByteNeo

March 7, 2023

Prepared by: xByteNeo Lead Security Researcher: - xByteNeo

Table of Contents

Protocol Summary

The PasswordStore protocol is a simple smart contract that allows a user to store and later retrieve a password. The contract is designed so that only the owner (the deployer) should be able to set and get the password. The password is stored as a private string variable within the contract.

Disclaimer

The xByteNeo 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 the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

- Solc version: 0.8.18
- Chain(s) to deploy to; Ethereum

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

The PasswordStore contract demonstrates common pitfalls in smart contract security, such as misunderstanding blockchain data privacy and missing access controls. The issues found are critical and undermine the intended functionality of the contract. We recommend a thorough redesign if password secrecy is required.

Issues found

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

During the audit, a total of 3 issues were identified: - 2 High severity issues: storing sensitive data on-chain in plaintext, and missing access control on the password setter. - 1 Informational issue: a documentation/natspec mismatch.

The High severity issues fundamentally break the intended security of the contract, allowing any user to read or overwrite the password. The Informational issue does not impact contract security but may cause confusion for developers or auditors.

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 only called by the owner of the contract.

We show one such method of reading any data off chain bellow.

Impact: Anyone can read the private password, severly 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 local running chain with anvil

```
make anvil
```

2. Deploy the contract to the local chain

```
make deploy
```

3. Run the command bellow to read the storage variable `s_password` content

Use parameter 1 to read the storage variable `s_password` and use the `--rpc-url` parameter to specify the local chain.

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

You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

Then you can decode the output to get the password.

```
cast parse-bytes32-string 0
    x6d7950617373776f72640000000000000000000000000000000000000000000014
```

You'll get the password as output:

`myPassword`

Recommended Mitigation: Due to this, the overall architecture of the contract should be re-evaluated. One possible solution is to encrypt the password off-chain, and only store the ecrypted version on-chain. This way even that the stored password still visible to anyone, it will be kept secret and only the ecryptor owner is able to decrypt it.

[H-2] The `PasswordStore::setPassword` has no access control, 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 overall purpose of the smart contract is that `This function allows only the owner to set a new password.`

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

Impact: Anyone can change the password, severly breaking the functionality of the protocol.

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

```
if(msg.sender != s.owner) {
    revert PasswordStore__NotOwner();
}
```

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exists causing the nastspec to be incorrect

Description: The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`.

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

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspect line.

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