Protocol Audit Report

Version 1.0

RealGC

December 28, 2024

Protocol Audit Report

RealGC

December 27, 2024

Prepared by: RealGC Lead Auditors: - RealGC

Table of Contents

Protocol Summary

PasswordStore is a smart contract designed to provide password storage functionality. The contract allows for storing and retrieving passwords, but contains significant security vulnerabilities that compromise its core functionality.

Disclaimer

The RealGC 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

	Impact			
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

The audit scope encompasses the complete codebase of the PasswordStore smart contract, with particular focus on: - Password storage mechanism - Access control implementation - Data privacy considerations

Roles

The contract defines the following key roles: - Contract Owner: Should be the only address authorized to set passwords - Users: Any address that can view the password

Executive Summary

During our audit, we identified two critical vulnerabilities: 1. Private variables are publicly accessible on-chain 2. The setPassword function lacks access control

Issues found

Severity	Number of Issues
High	2
Medium	0
Low	0
Informational	0

Severity	Number of Issues
Gas	0

Findings

High

[H-01] Private Variables Are Publicly Accessible On-Chain

Severity: High

- Impact: High - Password data is exposed
- Likelihood: High - Can be easily extracted by anyone

Description Despite using the `private` visibility modifier, the password stored in `PasswordStore::s_password` is publicly accessible due to blockchain's transparent nature. All data stored on-chain can be read by anyone, regardless of visibility modifiers.

Impact

- Complete compromise of password confidentiality
- Nullifies the contract's core security purpose
- Any user can read the stored password

Proof of Concept The following steps demonstrate password extraction:

1. Deploy contract and set password:

```
forge script script/DeployPasswordStore.s.sol --rpc-url http://
    localhost:8545 --broadcast
```

2. Extract password from storage slot 1:

```
cast storage <CONTRACT_ADDRESS> 1 --rpc-url http://127.0.0.1:8545
# Returns: 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

3. Decode the password:

```
cast --to-ascii 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
# Returns: myPassword
```

Recommendations

1. **Do Not Store Sensitive Data On-Chain**
    - Move sensitive data to secure off-chain storage
    - Use blockchain for access control and verification only
2. **Alternative Approaches**
    - Implement zero-knowledge proofs for password verification
    - Use commitment schemes or hash-based approaches
    - Store encrypted data if on-chain storage is required

[H-02] Missing Access Control in setPassword Function

Severity: High

- Impact: High - Unauthorized password modification
- Likelihood: High - No access control implemented

Description The `setPassword` function lacks proper access control, allowing any address to modify the stored password. This contradicts the contract's intended behavior where "only the owner should be able to set the password".

Vulnerable Code

```
function setPassword(string memory newPassword) external {
    // @audit No access control check
    s_password = newPassword;
    emit SetNewPassword();
}
```

Impact

- Any address can modify the stored password
- Complete compromise of access control
- Owner's privileged access is nullified

Proof of Concept

```
function test_setPassword_Success_WhenCalledByAnyone(address
    randomAddress) public {
     vm.assume(randomAddress != owner);
     vm.startPrank(randomAddress);
     string memory expectedPassword = "myNewPassword";
     passwordStore.setPassword(expectedPassword);
     string memory actualPassword = passwordStore.getPassword();
     assertEq(actualPassword, expectedPassword);
}
```

Test execution confirms the vulnerability:

```
forge test --match-test test_setPassword_Success_WhenCalledByAnyone
[PASS] test_setPassword_Success_WhenCalledByAnyone(address) (runs: 257)
```

Recommended Fix Implement proper access control:

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

Security Considerations

1. Ensure owner address is properly initialized and cannot be manipulated
2. Consider implementing a more robust access control system (e.g., OpenZeppelin's Ownable)
3. Add events for better transparency and monitoring

Medium

No medium severity issues found

Low

No low severity issues found

Informational

No informational issues found

Gas

No gas optimization issues found

Conclusion

The contract contains high severity security vulnerabilities that completely compromise its intended security properties. Immediate fixes are required before deployment.