Protocol Audit Report

KineticsOfWeb3

August 12, 2024

Prepared by: KineticsOfWeb3 Lead Security Researcher:

- KineticsOfWeb3

---

title: Protocol Audit Report author: KineticsOfWeb3 date: August 12, 2024

Table of Contents

Protocol Summary

The protocol summary you provided is well-structured and informative, but to align more closely with industry standards, particularly in smart contract development and blockchain protocols, it could be formatted and presented with a bit more emphasis on clarity, brevity, and critical details. Here's a revised approach that follows standard practices:

Protocol Summary: Secure Password Storage Smart Contract Purpose: This smart contract is designed to securely store and retrieve passwords on the blockchain, ensuring that only authorized users can access their passwords, while mitigating potential security vulnerabilities inherent to on-chain data storage.

Core Functionality:

Password Storage: Users can store their passwords securely on the blockchain. Password Retrieval: Only the user who stored the password can retrieve it. Access Control: Access to password management functions is restricted to the owner (the user who set the password). Security Analysis:

On-Chain Data Exposure (H-1):

Issue: Passwords, even marked as private, are publicly accessible on the blockchain. Mitigation: Encrypt passwords off-chain before storing them on-chain, and remove functions that might expose the data. Unauthorized Access (H-2):

Issue: The setPassword function lacks access control, allowing anyone to change the password. Mitigation: Implement an ownership check to ensure only the contract owner can modify the password. Naming Conventions:

Event Naming (L-1): Use descriptive event names (e.g., PasswordUpdated) to improve clarity and maintainability. Error Naming (i-1): Simplify error names (e.g., NotOwner) to reduce gas costs and enhance readability. Compiler Version Management (M-1):

Issue: The contract locks to a specific compiler version, which may lead to compatibility issues. Mitigation: Use a version range (e.g., ˆ0.8.18) to allow for updates while maintaining compatibility. Conclusion: The audit identifies critical and non-critical vulnerabilities in the protocol's password storage mechanism. While the protocol provides essential functionalities for secure password storage, implementing the recommended mitigations is crucial to ensuring the protocol's security and robustness. By addressing these issues, the protocol can better protect users' sensitive information and adhere to best practices in smart contract development.

Disclaimer

The KineticsOfWeb3 team makes all efforts to identify as many vulnerabilities as possible within the given time period but holds no responsibility 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 focused solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Finding ID	Description	Impact	Likelihood	Severity
H-1	On-Chain Visibility of "Private" Variables	High	High	High
H-2	Lack of Access Control on `setPassword` Function	Critical	High	Critical
L-1	Event Naming Convention	Low	Medium	Low
G-1	Error Naming Convention	Low	High	Low
M-1	Compiler Version Lock	Medium	Medium	Medium

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

Audit Details

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

`7d55682ddc4301a7b13ae9413095feffd9924566`

Scope

The audit focused on the security aspects of the Password Store smart contract, specifically evaluating the following components:

- `PasswordStore` contract's functions, state variables, and event handling.
- Access control mechanisms.
- Compliance with Solidity best practices.
- Gas efficiency of the contract.

Roles

-Owner: The user who has the authority to set and view the password. -Outsiders: No one else is allowed to read or set the password.

Executive Summary

```
The Password Store smart contract was audited with a focus on identifying potential security
```

Issues found

Issues Found High Severity: 2 issues Medium Severity: 1 issue Low Severity: 1 issue Informational: 1 issue Gas Efficiency Concerns: 1 issue Total: 6

Findings

High

[H-1] On-Chain Visibility of “Private” Variables: Exposing Sensitive Data and Compromising Protocol Security

Likelihood & Impact:

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

Description: All data stored on-chain is accessible to anyone, regardless of the Solidity visibility keyword used. In this case, the `PasswordStore::s_password` variable is intended to be private and only accessed through the `PasswordStore::getPassword` function, which is designed to be callable only by the contract owner. However, because blockchain data is publicly accessible, the password is not truly private.

Proof of Concept: The following steps demonstrate how anyone can retrieve the password directly from the blockchain:

Set up a Local Blockchain:

`bash`

make anvil

Deploy the Contract:

`bash`

make deploy

Read the Stored Password: Use the storage tool to fetch the password:

`bash`

cast storage 1 –rpc-url http://127.0.0.1:8545 The output will resemble:

0x6d7950617373776f7264000000000000000000000000000000000000000000014 Convert the Hexadecimal Value to a String:

`bash`

cast –parse-bytes32-string 0x6d7950617373776f7264000000000000000000000000000000000000000000014 This returns: mypassword

Recommended Mitigation: To address this vulnerability, the contract’s architecture should be reconsidered. One possible solution is to encrypt the password off-chain before storing it on-chain. This approach would require users to maintain an additional off-chain password for decryption. Additionally, it would be wise to remove the view function to prevent users from inadvertently exposing the password when making a transaction.

[H-2] Lack of Access Control on setPassword Function (Missing Ownership Check + Unauthorized Access)

Likelihood & Impact:

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

Description: The `setPassword` function in the `PasswordStore` contract lacks proper access control. Currently, the function is external, which means anyone can call this function and set the password to a new value without any restrictions.

Impact: This issue allows any external party to change the stored password without permission, which could lead to unauthorized access or loss of integrity of the stored data. The lack of access control undermines the contract's security by exposing critical functionality to unauthorized users.

Proof of Concept:

```
// Current vulnerable function in the contract:
function setPassword(string memory newPassword) external {
    s_password = newPassword;
    emit SetN``etPassword();
}

// Any external address can call this function:
address attacker = 0xAttackerAddress;
attacker.call(abi.encodeWithSignature("setPassword(string)", "newMaliciousPassword"));
```

Recommended Mitigation: Implement an access control mechanism by adding a modifier to check if msg.sender is the owner of the contract before allowing the password to be set. This ensures that only the owner can modify the password.

```
// Access control modifier to restrict function to owner only
modifier onlyOwner() {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
    _;
}

// Updated setPassword function with access control
function setPassword(string memory newPassword) external onlyOwner {
s_password = newPassword;
emit SetNetPassword();
}
```

Medium

[M-1] Compiler Version (Locked Version + Potential Compatibility Issues)

Likelihood & Impact:

-Impact: MEDIUM -Likelihood: MEDIUM -Severity: MEDIUM

Description: The contract uses pragma solidity 0.8.18; which locks the contract to a specific compiler version. This approach could lead to issues if bug fixes or optimizations are introduced in later versions of the compiler.

Impact: Locking the compiler version can prevent the contract from benefiting from future compiler improvements and may also introduce compatibility issues if other contracts or libraries use different versions.

Proof of Concept:

```
// Current pragma directive:
pragma solidity 0.8.18;
```

Recommended Mitigation: Consider using a version range like ˆ0.8.18 to allow for patch-level updates while maintaining compatibility.

```
// Improved pragma directive:
pragma solidity ^0.8.18;
```

Low

[i -1] Error Naming Convention (Unnecessary Complexity + Increased Gas Costs)

Likelihood & Impact:

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

Description: The error `PasswordStore__NotOwner` in the `PasswordStore` contract includes double underscores, which are unnecessary and could lead to slightly increased gas costs due to the longer identifier.

Impact: Longer error names consume more gas and make the code less readable. Simplifying the error name can save gas and improve code clarity.

Proof of Concept:

```
// Current error declaration:
error PasswordStore__NotOwner();
```

Recommended Mitigation: Simplify the error name to something like NotOwner or Unauthorized to reduce gas costs and improve readability.

```solidity
// Improved error declaration:
error NotOwner();
```

Gas

[**i -1**] **Error Naming Convention (Unnecessary Complexity + Increased Gas Costs)**

Likelihood & Impact:

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

Description: The error `PasswordStore__NotOwner` in the `PasswordStore` contract includes double underscores, which are unnecessary and could lead to slightly increased gas costs due to the longer identifier.

Impact: Longer error names consume more gas and make the code less readable. Simplifying the error name can save gas and improve code clarity.

Proof of Concept:

```solidity
// Current error declaration:
error PasswordStore__NotOwner();
```

Recommended Mitigation: Simplify the error name to something like NotOwner or Unauthorized to reduce gas costs and improve readability.

```solidity
// Improved error declaration:
error NotOwner();
```