Smart Contract Audit Report

hello_blockchain

Generated on 5/26/2025

AUDIT INFORMATION

Contract Name:	hello_blockchain
Audit Date:	5/26/2025
Auditor:	AuditWarp AI (powered by Google Gemini)
Report Type:	Automated Security Analysis

CERTIFIED BY
AUDITWARP

DISCLAIMER

This audit report is provided as-is without any warranties. While we strive for accuracy, this automated analysis may not capture all potential vulnerabilities. Always conduct additional security reviews before deploying in production.

AUDIT FINDINGS

EXECUTIVE SUMMARY

Move Smart Contract Security Audit Report: `hello_blockchain::message` **Certified by AuditWarp** ## Executive Summary This report analyzes the Move smart contract located in the `hello_blockchain::message` module for potential vulnerabilities.

SECURITY ASSESSMENT

} ``` * **Severity:** Low 3.

IDENTIFIED VULNERABILITIES

- **Vulnerability Score: 3/10** ## Summary of Risks * **Low Impact:** The risk associated with the reported issues are low, primarily focusing on gas optimization, event emission completeness, and a minor griefing opportunity.
- This is primarily an informational issue.

RECOMMENDATIONS

- * **Location:** `hello_blockchain::message::set_message` * **Code Snippet:** ```move event::emit(MessageChange { account: account_addr, from_message, to_message: copy message, // Unnecessary copy }); ``` * **Severity:** Informational ## Recommendations 1.
- **Consider String Length Limit:** To mitigate the potential string resource exhaustion griefing vector, consider adding a maximum length limit to the `message` parameter in the `set_message` function.
- This limit should be reasonable for typical use cases.

DETAILED ANALYSIS

1. The contract allows users to set and retrieve a message associated with their account. While the code is relatively simple, it's crucial to review even basic contracts for potential oversights. The

primary findings relate to event emission best practices and potential griefing vectors related to resource management.

2. The contract's core functionality (setting and retrieving messages) appears secure. ## Detailed Findings 1. **Missing Event Emission on Initial Message Set (Informational)** * **Description:** The `set_message` function only emits a `MessageChange` event when the `MessageHolder` resource already exists for the user.

3. No event is emitted when the resource is first created and the initial message is set. * **Impact:** Low. Clients relying on `MessageChange` events will not be notified when an account initially sets their message.

4. This can lead to incomplete event history on frontends or data analysis pipelines. **Potential Griefing Vector: String Resource Exhaustion (Low)** * **Description:** An attacker could potentially grief other users by repeatedly setting extremely long strings as their message. This could lead to excessive gas consumption for those users when they subsequently update their message (due to the string copying and event emission).

5. * **Impact:** Low. While a griefing attack is possible, the impact is limited. The attacker is essentially paying for the cost of setting the large string.

6. However, the cost for other users updating their message will be increased. * **Location:** `hello_blockchain::message::set_message` * **Code Snippet:** ```move public entry fun set_message(account: signer, message: string::String) acquires MessageHolder { // ... let old_message_holder = borrow_global_mut<MessageHolder>(account_addr); let from_message = old_message_holder.message; // ...

7. **Unnecessary `copy` in Event Emission (Gas Optimization, Informational)** * **Description:** The `copy message` in the event emission creates an unnecessary copy of the string. This incurs additional gas costs. * **Impact:** Low.

8. This is a gas optimization opportunity. Removing the `copy` will slightly reduce the gas cost of updating the message. **Emit Event on Initial Message Set:** Modify the `set_message` function to emit a `MessageChange` event even when the `MessageHolder` resource is first created.

9. The `from_message` field can be an empty string or a special "initial" value to indicate the creation of the resource. ```move public entry fun set_message(account: signer, message: string::String) acquires MessageHolder { let account_addr = signer::address_of(&account); if (!exists<MessageHolder>(account_addr)) { event::emit(MessageChange { account: account_addr, from_message: string::utf8(b""), // Or another special value to_message: copy message, }); move_to(&account, MessageHolder { message, }) } else { let old_message_holder = borrow_global_mut<MessageHolder>(account_addr); let from_message = old_message_holder.message; event::emit(MessageChange { account: account_addr, from_message, to_message: copy message, }); old_message_holder.message = message; } } ``` 2. ```move const MAX_MESSAGE_LENGTH: u64 = 256; // Example limit public entry fun set_message(account: signer, message: string::String) acquires MessageHolder { assert!(string::length(&message) <= MAX_MESSAGE_LENGTH, error::invalid_argument(100)); //Example error code // ...

10. rest of the function ... **Remove Unnecessary `copy`:** Remove the `copy` keyword when emitting the `to_message` in the `MessageChange` event to reduce gas consumption. ```move event::emit(MessageChange { account: account_addr, from_message, to_message: message, }); ```.

CONTRACT CODE SUMMARY

Contract Source Code:

```
module hello_blockchain::message {
    use std::error;
    use std::signer;
    use std::string;
    use aptos_framework::event;
    //:!:>resource
    struct MessageHolder has key {
        message: string::String,
    }
    //<:!:resource
    #[event]
    struct MessageChange has drop, store {
        account: address,
        from_message: string::String,
        to_message: string::String,
    }
    /// There is no message present
    const ENO_MESSAGE: u64 = 0;
    #[view]
    public fun get_message(addr: address): string::String acquires MessageHolder {
        assert!(exists<MessageHolder>(addr), error::not_found(ENO_MESSAGE));
        borrow_global<MessageHolder>(addr).message
    }
    public entry fun set_message(account: signer, message: string::String)
    acquires MessageHolder {
        let account_addr = signer::address_of(&account);
        if (!exists<MessageHolder>(account_addr)) {
            move_to(&account, MessageHolder {
                message,
            })
        } else {
            let old_message_holder = borrow_global_mut<MessageHolder>(account_addr);
            let from_message = old_message_holder.message;
            event::emit(MessageChange {
                account: account_addr,
                from_message,
                to_message: copy message,
            });
            old_message_holder.message = message;
        }
    }
    #[test(account = @0x1)]
    public entry fun sender_can_set_message(account: signer) acquires MessageHolder {
        let addr = signer::address_of(&account);
        aptos_framework::account::create_account_for_test(addr);
        set_message(account, string::utf8(b"Hello, Blockchain"));
        assert!(
            get_message(addr) == string::utf8(b"Hello, Blockchain"),
```

Displaying 48 of 52 lines

[Code truncated for PDF - full code analyzed in audit]