Audit

WALC engaged Mario Reder as an independend Near Protocol expert to audit their fungible token smart contract, which is deployed on the address walc.near.

Table of contents

1. About the auditor

Mario Reder is an independent Near Protocol expert with 2+ years of experience developing smart contracts on Near and 5+ years of experience with the Rust programming language and WebAssembly.

Contact:

- Email: mario.reder@pm.me
- Github: Tarnadas
- Telegram: marior_dev

2. Scope

WALC fungible token contract on Github.

3. Test approach & methodology

Vulnerabilities or issues observed are ranked based on the risk assessment methodology by measuring the LIKELIHOOD of a security incident and the IMPACT should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

```
5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.
```

RISK SCALE - IMPACT

```
5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.
```

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

Value	Severity
10	CRITICAL
9 - 8	HIGH
7 - 6	MEDIUM
5 - 4	LOW
3 - 1	VERY LOW AND INFORMATIONAL

The test approach & methodology used in this audit is similar to Halborn.

4. Overview

Security Analysis	Risk Level
Fungible token transfer fails to send unused gas	Low
Missing possibility to unregister account	Informational
Usage of vulnerable crates	Informational

5. Findings

5.1 (VULN-01) Fungible token transfer fails to send unused gas - Low

A call to `ft_transfer_call` fails to attach unused gas to `ft_on_transfer` function. Instead it only attaches a fixed amount of gas. This can likely result in the receiving smart contract to run out of gas, resulting in the receipt to fail. A proper refund will be done in the `ft_resolve_transfer` function.

```
#[payable]
fn ft_transfer_call(
    &mut self,
    receiver_id: AccountId,
    amount: U128,
    memo: Option<String>,
    msg: String,
) -> PromiseOrValue<U128> {
    // ...

    ext_ft_receiver::ext(receiver_id.clone())
        .with_static_gas(GAS_FOR_FT_TRANSFER_CALL)
        .ft_on_transfer(sender_id.clone(), amount.into(), msg)
        .then(
            ext_ft_resolver::ext(env::current_account_id())
```

```
                .with_static_gas(GAS_FOR_RESOLVE_TRANSFER)
                .ft_resolve_transfer(sender_id, receiver_id, amount.into()),
        )
        .into()
}
```

Risk Level

Likelihood - 4

Impact - 1

Recommendation

In order for the smart contracts who receive the fungible token to properly execute, it is necessary to attach any unused gas. Otherwise this could result in several applications like Decentralized Exchanges (DEXes) or money markets being (partially) unusable. The amount of gas that can be attached to the cross contract call can be calculated by getting the unused gas via `env::unused_gas()` and subtracting a fixed amount of gas that would be necessary to finish remaining execution plus the gas that is attached to `ft_resolve_transfer`.

5.2 (VULN-02) Missing possibility to unregister account - Informational

The NEP-145 standard defines Storage Management and includes a function `storage_unregister` to unregister an account. This function is missing in the WALC fungible token smart contract.

Additionally it needs to be noted that if such a function will be added to the smart contract, then this will result in another smart contract bug inside the `ft_resolve_transfer` function, where it would be possible to unregister an account that would receive a refund. The receipt would panic in this case, resulting in the fungible tokens staying inside the `receiver_id` 's wallet instead of being burnt.

```
#[private]
pub fn ft_resolve_transfer(
    &mut self,
    sender_id: &AccountId,
    receiver_id: AccountId,
    amount: U128,
) -> U128 {
    let amount: Balance = amount.into();

    let unused_amount = match env::promise_result(0) {
        // ...
    };

    if unused_amount > 0 {
        // ...
        if receiver_balance > 0 {
            // ...

            self.internal_transfer(&receiver_id, &sender_id, refund_amount, Some("Refund".to_string()));

            // ...
        }
    }

    amount.into()
}

pub(crate) fn internal_transfer(
    &mut self,
    sender_id: &AccountId,
    receiver_id: &AccountId,
    amount: Balance,
```

```
    memo: Option<String>,
) {
    // ...
    self.internal_deposit(receiver_id, amount);
    // ...
}

pub(crate) fn internal_deposit(&mut self, account_id: &AccountId, amount: Balance) {
    // Get the current balance of the account. If they're not registered, panic.
    let balance = self.internal_unwrap_balance_of(account_id);
    // ...
}
```

Risk Level

Likelihood - 3

Impact - 1

Recommendation

It is recommended to be compliant with the NEP-145 standard and implement the missing function. The `ft_resolve_transfer` function also needs changes to properly check whether the account receiving the refund has been unregistered and do a fungible token burn in this case.

5.3 (VULN-03) Usage of vulnerable crates - Informational

ID	package	description
RUSTSEC-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed25519-dalek`

Risk Level

Likelihood - 2

Impact - 1

Recommendation

The vulnerable crates cannot impact the underlying application, but it is still advised to be aware of them and attempt to update them to a novulnerable version. Furthermore, it is necessary to set up dependency monitoring to always be alerted when a new vulnerability is disclosed in one of the project's crates.