RECONWITHME

SECURITY AUDIT REPORT FOR

WAPAL

July 18, 2024

Executive Summary

This smart contract audit report is prepared for *candymachine.move, bucket_table.move, bit_vector.move, merkle_proof.move*, smart contracts of *Wapal*, source code review exercise.

Type
Smart Contract

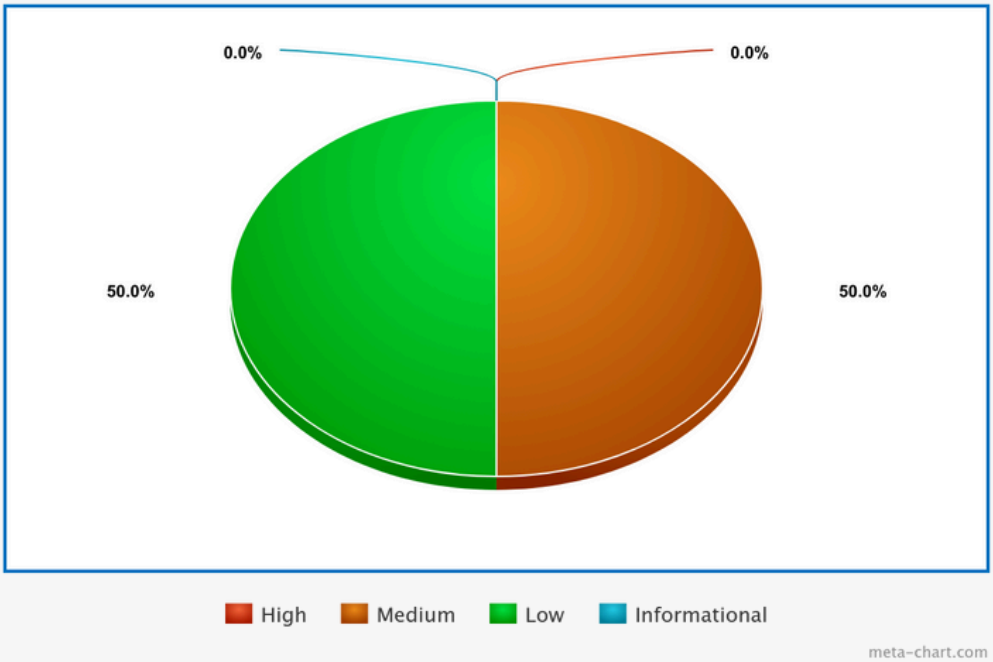
Languages
Move

Timeline
July 1, 2024 to July 18, 2024

Methodology
White Box Testing

Change log

July 1, 2024 - Requirement received via github - commit history - 982039bf9fa7363dc047e8befd1c0d03f50106d3
July 1, 2024 - Beginning of first phase of audit.
July 18, 2024 - Completion of first phase of audit, report submission.



High Risk Vulnerabilities that can be exploited publicly, workaround or fix/ patch available by the vendor.	0	**Low Risk** Vulnerabilities may not have public exploit (code) available or cannot be exploited in the wild. Vulnerability observed may not have a high rate of occurrence. Patch/ workaround released by the vendor.	1
Medium Risk Vulnerabilities may not have public exploit (code) available or cannot be exploited in the wild. Patch/ workaround not yet released.	1	**Informational Risk** Vulnerabilities that have the minimum impact on the system.	0
Undetermined Risk Vulnerabilities that has uncertain impact.	0		

Summary of Findings

Vulnerability	Risk	Status
1. Precision error: Mokshya fee can be 0	Medium	Open
2. Public sale time can be before Pre-sale time	Low	Open

Objective

The objective of the security audit of smart contract of *Wapal* was to assess the state of security and uncover vulnerabilities in the smart contract and provide with a detail report comprising remediation strategy and recommendations to help mitigate the identified vulnerabilities and risks during the activity.

The assessment was conducted using two methods; source code review and black box testing.

Scope & Requirements

The technical scope for the conducted application security testing activity was restricted to:

- bit_vector.move
- bucket_table.move
- candymachine.move
- merkle_proof.move

Files url

https://github.com/mokshyaprotocol/aptos-nft-random-mint-tokenv2
Git commit hash - 982039bf9fa7363dc047e8befd1c0d03f50106d3

Provided Date

July 01, 2024; Email Access given via Github.

Methodology

White Box Testing/Source Code Review

Understand Program Specification

Understanding programming language used and coding practices followed is a key in performing efficient source code review solution. Our security researchers begin source code review by first understanding the program specification.

Obtain and Review Source Code

Our security researchers then co-ordinate with the development team to obtain the source code and start reviewing the codes line by line to identify flaws.

Indentify Flaws

We apply a rigorous approach to identify flaws. That is we adhere to strict standard practices and retest major functions again to make sure we don't miss a single loophole.

Reporting

A documentation of where the flaws has been found and how the patches can be done will be submitted to the development team for a fix.

Findings from the assessment

We found the following findings after performing the security audit of the smart contracts of *Wapal* through source code review.

Findings from Manual Testing

1. Precision error: Mokshya fee can be 0

Description

Fees cultivated by Moksyha are calculated with the following equation:
fee = (MINT_FEE_NUMERATOR * mint_price) / MINT_FEE_DENOMINATOR;
Where MINT_FEE_NUMERATOR = 300 and MINT_FEE_DENOMINATOR = 10000

Given a small enough mint_price, it is possible for the value of fee to lie between 1 and 0, e.g. 0.5

However, due to the limitation the move language, the digits after the decimal points a truncated, leading to the value of Fee to be rounded to 0.

Impact

High

Risk	Status
Medium	Open

Proof of Concept

Suppose the owner of the collection set mint_price to 0.0000001 APT, this can be done via the dAPP.

Upon fees calculation, with the following values:

let fee = (MINT_FEE_NUMERATOR * mint_price) / MINT_FEE_DENOMINATOR;
= (300 * (0.0000001 * 10^8)) / 10000

we get fee = 0.3
Here the digits after the decimal point is truncated, leading the value of fee to be 0.

sources/candymachine.move:L#423

```
411                 creator,
412                 candy_data.collection_name,
413                 candy_data.collection_description,
414                 token_name,
415                 baseuri,
416                 vector::empty<String>(),
417                 vector::empty<String>(),
418                 vector::empty()
419             );
420             object::transfer(creator, minted_token, receiver_addr);
421
422             // Split sale price into Mokshya fee and the collection owner
423             let fee = (MINT_FEE_NUMERATOR * mint_price) / MINT_FEE_DENOMINATOR;
424             let collection_owner_price = mint_price - fee;
425             coin::transfer<AptosCoin>(receiver, MokshyaFee, fee);
426             coin::transfer<AptosCoin>(receiver, candy_admin, collection_owner_price);
427
428             // Increment the total mints for the collection
429             candy_data.minted = candy_data.minted + 1;
430
431             // Increment the total mints for the contract!
432             mint_data.total_mints = mint_data.total_mints + 1
```

Remediation

It is recommended to:

- Set a minimum mint price ensuring that the fee will never round down to zero.
- Check that fees are non-zero and handle the situation specifically, for example by set a minimum fee or rejecting the transaction.

2. Public sale time can be before Pre-sale time

Description

The function update_public_sale_time() only has checks to ensure the provided time is at some point in the future, but lacks the check to see if the provided time is after the pre-sale time has passed. This makes it possible to update Public sale time to a time before the Pre-sale time.

Furthermore, the update_wl_sale_time() which updates the pre-sale time is also lacking a check to ensure the new provided time is before the public sale time.

Impact

Unknown

Risk	Status
Low	Open

Proof of Concept

sources/candymachine.move:L#468-L#491

```
467         /// Update whitelist / presale sale time, only creator of mint can do this, and it must be in the future
468         public entry fun update_wl_sale_time(
469             account: &signer,
470             candy_obj: address,
471             presale_mint_time: u64
472         ) acquires CandyMachine, ResourceInfo {
473             let account_addr = signer::address_of(account);
474             let resource_data = borrow_global<ResourceInfo>(candy_obj);
475             let now = aptos_framework::timestamp::now_seconds();
476             assert!(resource_data.source == account_addr, EINVALID_SIGNER);
477 
478             let candy_data = borrow_global_mut<CandyMachine>(candy_obj);
479             assert!(presale_mint_time >= now, EINVALID_MINT_TIME);
480             candy_data.presale_mint_time = presale_mint_time;
481         }
482 
483         /// Update public sale time, only creator of mint can do this, and it must be in the future
484         public entry fun update_public_sale_time(
485             account: &signer,
486             candy_obj: address,
487             public_sale_mint_time: u64
488         ) acquires CandyMachine, ResourceInfo {
489             let account_addr = signer::address_of(account);
490             let resource_data = borrow_global<ResourceInfo>(candy_obj);
491             let now = aptos_framework::timestamp::now_seconds();
492             assert!(resource_data.source == account_addr, EINVALID_SIGNER);
493 
494             let candy_data = borrow_global_mut<CandyMachine>(candy_obj);
495             assert!(public_sale_mint_time >= now, EINVALID_MINT_TIME);
496             candy_data.public_sale_mint_time = public_sale_mint_time;
```

Remediation

Add a check to ensure the presale time does not exceed the public sale time and vise versa.

For example:

```
public entry fun update_wl_sale_time(
    account: &signer,
    candy_obj: address,
    presale_mint_time: u64
) acquires CandyMachine, ResourceInfo {
    let account_addr = signer::address_of(account);
    let resource_data = borrow_global<ResourceInfo>(candy_obj);
    let now = aptos_framework::timestamp::now_seconds();
    assert!(resource_data.source == account_addr, EINVALID_SIGNER);

    let candy_data = borrow_global_mut<CandyMachine>(candy_obj);
    assert!(presale_mint_time >= now, EINVALID_MINT_TIME);
    assert!(presale_mint_time < candy_data.public_sale_mint_time);
    candy_data.presale_mint_time = presale_mint_time;
}

public entry fun update_public_sale_time(
    account: &signer,
    candy_obj: address,
    public_sale_mint_time: u64
) acquires CandyMachine, ResourceInfo {
    let account_addr = signer::address_of(account);
    let resource_data = borrow_global<ResourceInfo>(candy_obj);
    let now = aptos_framework::timestamp::now_seconds();
    assert!(resource_data.source == account_addr, EINVALID_SIGNER);

    let candy_data = borrow_global_mut<CandyMachine>(candy_obj);
    assert!(public_sale_mint_time >= now, EINVALID_MINT_TIME);
    assert!(candy_data.presale_mint_time < public_sale_mint_time);
    candy_data.public_sale_mint_time = public_sale_mint_time;
}
```

Conclusion

We found a total of 2 vulnerabilities in *candymachine.move, bucket_table.move, bit_vector.move, merkle_proof.move,* out of which 1 possessed medium risk, and 1 possessed low risk.

In addition to this audit, we encourage the team to organize a public vulnerability disclosure program and adopt other necessary measures to minimize cyber risks associated with smart contracts.

Disclaimer

Considering the evolving nature of risks and vulnerabilities associated with solidity language, smart contracts and Etherium network, ReconwithMe cannot wholly guarantee that the contract will be free of vulnerabilities after the audit. We encourage the team to organize a public vulnerability disclosure program and adopt other necessary measures to minimize cyber risks associated with smart contracts.

About ReconwithMe

ReconwithMe is a cyber security firm dedicated to providing vulnerability management solutions. ReconwithMe has been trusted by industries ranging from fintech to blockchain and SAAS to Ecommerce. With cyber attacks rising rapidly, businesses require cyber security solutions more than ever. Our tailor-made vulnerability management solutions help prevent cyber attacks. Our team consists of self-taught professional cyber security practitioners who follow international standard security protocols and possess strong work ethics. Our team has been recognized by tech giants such as Facebook, Microsoft, Sony, Etsy and others for contributing to make them more secure.

ReconwithMe Pvt. Ltd.
Address - Kupondole, Lalitpur, Nepal
Email - support@ReconwithMe.io