MOVEBIT
Securing the Move Ecosystem

Mokshya/Wapal Aptos NFT Mint Smart Contract **Audit Report**

https://twitter.com/movebit_

contact@movebit.xyz

Mokshya/Wapal Aptos NFT Mint Smart Contract Audit

1 Executive Summary

1.1 Project Information

Type	NFT
Auditors	MoveBit
Timeline	2023–03–03 to 2023–03–09
Languages	Move
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	Repository: https://github.com/mokshyaprotocol/aptos-candymachine Received Commit: 14c9e6e56041cfea3bd025fc056b3ca4cb410d43 Last Reviewed Commit: 353e25a98726f5af97fdd638f17e1a88cc16b3d0

1.2 Issue Statistic

Item	Count	Fixed	Pending	Confirmed
Total	5	5		
Minor	3	3		
Medium	2	2		

Major				
Critical				

1.3 Issue Level

- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

1.4 Issue Status

- **Fixed:** The issue has been resolved.
- **Pending:** The issue has been acknowledged by the code owner, but has not yet been resolved. The code owner may take action to fix it in the future.
- **Confirmed:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

2 Summary of Findings

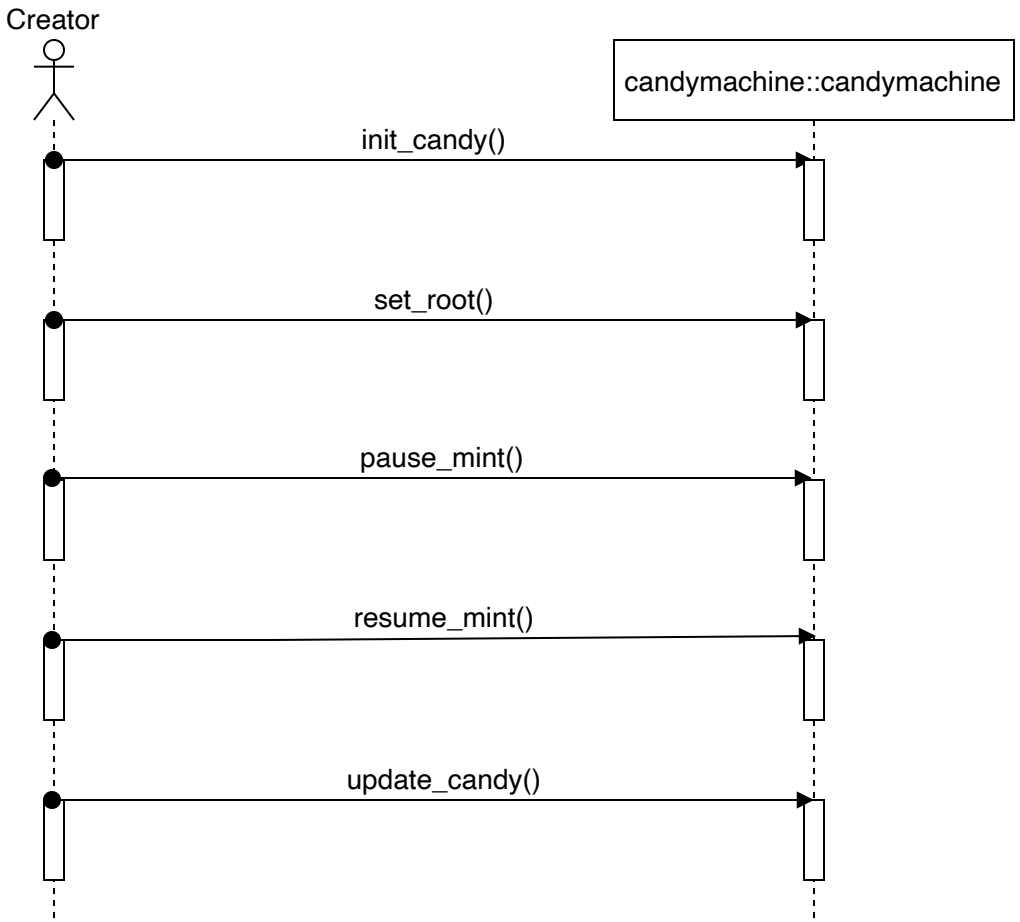
Mokshya is an open-source protocol building smart contacts and SDKs for Aptos Blockchain. The objective is to help onboard the next 10,000+ web3 projects by building well-documented smart contracts and SDKs for developers. Our team mainly focused on reviewing the Code Security and normative. During the testing process, our team also maintains close communication with the project team to ensure that we have a correct understanding of business requirements. As a result, our team found a total of 5 issues. The team discussed these issues together and communicated with the development team.

3 Participant Process

Here are the relevant actors with their respective abilities within the `Mobius-NFT-AMM` Smart Contract：

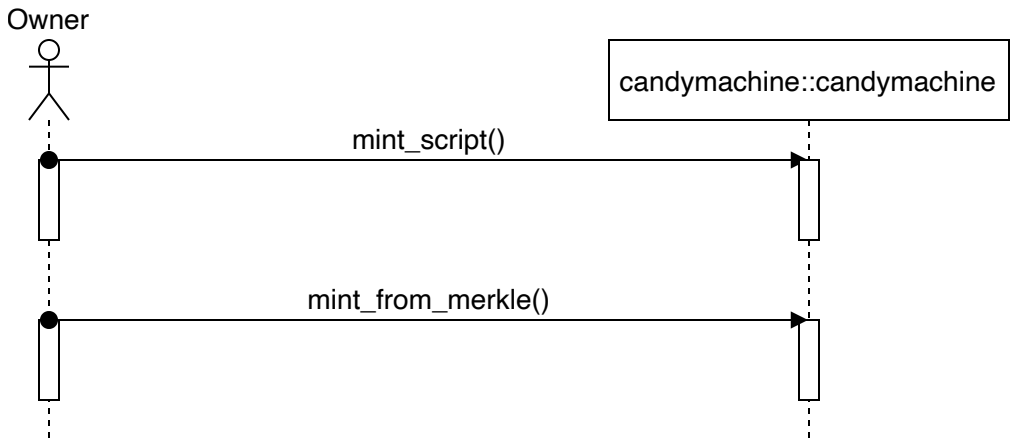
Creator

- Creators can create collections and initialize configuration parameters.
- Creators can reset `merkle_root`.
- Creators can pause mint.
- Creators can resume mint.
- Creators can update parameters.



Owner

- Owner can mint.
- Owner can mint according to Merkle.



4 MoveBit Audit BreakDown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values

- The flow of capability
- Witness Type

5 Methodology

The security team adopted the "**Testing and Automated Analysis**", "**Code Review**" and "**Formal Verification**" strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

Code scope sees **Appendix 1**.

(3) Formal Verification

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

6 Findings

6.1 Wrong condition in `assert`

Severity: Medium

Status: Fixed

Descriptions: In the function `update_candy`, the parameter `'royalty_points_denominator'` judges the wrong condition here, which will lead to never being able to update `candy_data.royalty_points_denominator`.

Code Location: sources/candymachine.move, line 308.

candymachine.move

```
public entry fun update_candy(
    account: &signer,
    candymachine: address,
    royalty_points_denominator: u64,
    royalty_points_numerator: u64,
    presale_mint_time: u64,
    public_sale_mint_price: u64,
    presale_mint_price: u64,
    public_sale_mint_time: u64,
)acquires ResourceInfo,CandyMachine{
    ...
    assert!(royalty_points_denominator == 0, EINVALID_ROYALTY_NUMERATOR_DENOMINATOR);
    if (royalty_points_denominator>0){
        candy_data.royalty_points_denominator = royalty_points_denominator
    };
    ...
}
```

Suggestion: The if condition below has judged `'royalty_points_denominator'`, directly delete this assertion `assert!(royalty_points_denominator == 0, EINVALID_ROYALTY_NUMERATOR_DENOMINATOR);`.

6.2 Unverified `public_sale_mint_time` must be greater than `presale_mint_time`

Severity: Medium

Status: Fixed

Descriptions: In the function `candymachine::mint_from_merkle`, it is known that `public_sale_mint_time` must be greater than `presale_mint_time`, but this is not verified when creating and modifying `CandyMachine`. If `public_sale_mint_time` is less than or equal to `presale_mint_time` when creating `CandyMachine`, whitelist users will not be able to mint.

Code Location: sources/candymachine.move, lines 72, 292.

Suggestion: Add `assert` to the functions `candymachine::init_candy` and `candymachine::update_candy` to verify that `public_sale_mint_time` must be greater than `presale_mint_time`.

candymachine.move

```
public entry fun init_candy(
    account: &signer,
    collection_name: String,
    collection_description: String,
    baseuri: String,
    royalty_payee_address:address,
    royalty_points_denominator: u64,
    royalty_points_numerator: u64,
    presale_mint_time: u64,
    public_sale_mint_time: u64,
    presale_mint_price: u64,
    public_sale_mint_price: u64,
    total_supply:u64,
    collection_mutate_setting:vector<bool>,
    token_mutate_setting:vector<bool>,
    public_mint_limit: u64,
    merkle_root: vector<u8>,
    seeds: vector<u8>
){
    ......
    assert!(public_sale_mint_time > presale_mint_time && presale_mint_time >= now,EINVALID_MINT_TIME);
    ......
}
```

6.3 Inappropriate `borrow`

Severity: Minor

Status: Fixed

Descriptions: In the function `candymachine::mint_script`, the resource `CandyMachine` is obtained through `borrow_global_mut`, but there is no need to modify `CandyMachine` in this function. Using `borrow_global_mut` may be risky, and the function `candymachine::mint_from_merkle` also has this problem. The same problem is similar to using `table_with_length::borrow_mut` in the function `candymachine::bucket_table::borrow`.

Code Location: sources/candymachine.move, line 133, 148, sources/bucket_table.move, line 149.

candymachine.move

```
public entry fun mint_script(
    receiver: &signer,
    candymachine: address,
)acquires ResourceInfo, CandyMachine,MintData,PublicMinters{
    let candy_data = borrow_global_mut<CandyMachine>(candymachine);
    let mint_price = candy_data.public_sale_mint_price;
    let now = aptos_framework::timestamp::now_seconds();
    assert!(now > candy_data.public_sale_mint_time, ESALE_NOT_STARTED);
    mint(receiver,candymachine,mint_price)
}
```

bucket_table.move

```
public fun borrow<K: copy + drop, V>(map: &mut BucketTable<K, V>, key: K): &V
{
    let index = bucket_index(map.level, map.num_buckets, sip_hash_from_value(&
key));
    let bucket = table_with_length::borrow_mut(&mut map.buckets, index);
    ......
}
```

Suggestion: Replace `borrow_global_mut` with `borrow_global`.

candymachine.move

```
public entry fun mint_script(
    receiver: &signer,
    candymachine: address,
)acquires ResourceInfo, CandyMachine,MintData,PublicMinters{
    let candy_data = borrow_global<CandyMachine>(candymachine);
    let mint_price = candy_data.public_sale_mint_price;
    let now = aptos_framework::timestamp::now_seconds();
    assert!(now > candy_data.public_sale_mint_time, ESALE_NOT_STARTED);
    mint(receiver,candymachine,mint_price)
}
```

Replace `table_with_length::borrow_mut` with `table_with_length::borrow`.

bucket_table.move

```
public fun borrow<K: copy + drop, V>(map: &BucketTable<K, V>, key: K): &V {
    let index = bucket_index(map.level, map.num_buckets, sip_hash_from_value(&
key));
    let bucket = table_with_length::borrow(&map.buckets, index);
    ......
}
```

6.4 The verification conditions of `assert` and `if` are repeated

Severity: Minor

Status: Fixed

Descriptions: In the function `candymachine::mint_from_merkle`, use `assert` to verify that `is_whitelist_mint` is true to continue to execute the code, and then repeat the judgment through `if`.

In the function `candymachine::update_candy`, `assert` has been used to verify that `public_sale_mint_time >= now && presale_mint_time >= now`, and the judgment is repeated through `if` below.

Code Location: sources/candymachine.move, line 159, 305.

candymachine.move

```
public entry fun mint_from_merkle(
    receiver: &signer,
    candymachine: address,
    proof: vector<vector<u8>>,
    mint_limit: u64
) acquires ResourceInfo,MintData,PublicMinters,CandyMachine,Whitelist{
    ......
    assert!(is_whitelist_mint, WhitelistMintNotEnabled);
    if(is_whitelist_mint){
        // No need to check limit if mint limit = 0, this means the minter ca
n mint unlimited amount of tokens
        if(mint_limit != 0){
            let whitelist_data = borrow_global_mut<Whitelist>(candymachine);
            if (!bucket_table::contains(&whitelist_data.minters, &receiver_add
r)) {
                // First time minting mint limit = 0
                bucket_table::add(&mut whitelist_data.minters, receiver_addr,
0);
            };
            let minted_nft = bucket_table::borrow_mut(&mut whitelist_data.mint
ers, receiver_addr);
            assert!(*minted_nft != mint_limit, MINT_LIMIT_EXCEED);
            *minted_nft = *minted_nft + 1;
            mint_data.total_apt=candy_data.presale_mint_price;
            mint_price = candy_data.presale_mint_price
        };
        mint(receiver,candymachine,mint_price)
    };
}

public entry fun update_candy(
    account: &signer,
    candymachine: address,
    royalty_points_denominator: u64,
    royalty_points_numerator: u64,
    presale_mint_time: u64,
    public_sale_mint_price: u64,
    presale_mint_price: u64,
    public_sale_mint_time: u64,
)acquires ResourceInfo,CandyMachine{
    let account_addr = signer::address_of(account);
    let resource_data = borrow_global<ResourceInfo>(candymachine);
    let now = aptos_framework::timestamp::now_seconds();
    assert!(public_sale_mint_time >=  now && presale_mint_time >= now, EINVALI
D_MINT_TIME);
```

```
    ...
    if (presale_mint_time>0){
        candy_data.presale_mint_time = presale_mint_time
    };
    if (public_sale_mint_time>0){
        candy_data.public_sale_mint_time = public_sale_mint_time
    };
    ...
}
```

Suggestion: Remove the `if` judgment.

candymachine.move

```
public entry fun mint_from_merkle(
    receiver: &signer,
    candymachine: address,
    proof: vector<vector<u8>>,
    mint_limit: u64
) acquires ResourceInfo,MintData,PublicMinters,CandyMachine,Whitelist{
    ......
    assert!(is_whitelist_mint, WhitelistMintNotEnabled);
    // No need to check limit if mint limit = 0, this means the minter can min
t unlimited amount of tokens
    if(mint_limit != 0){
        let whitelist_data = borrow_global_mut<Whitelist>(candymachine);
        if (!bucket_table::contains(&whitelist_data.minters, &receiver_addr))
{
            // First time minting mint limit = 0
            bucket_table::add(&mut whitelist_data.minters, receiver_addr, 0);
        };
        let minted_nft = bucket_table::borrow_mut(&mut whitelist_data.minters,
 receiver_addr);
        assert!(*minted_nft != mint_limit, MINT_LIMIT_EXCEED);
        *minted_nft = *minted_nft + 1;
        mint_data.total_apt=candy_data.presale_mint_price;
        mint_price = candy_data.presale_mint_price
    };
    mint(receiver,candymachine,mint_price)
}

public entry fun update_candy(
    account: &signer,
    candymachine: address,
    royalty_points_denominator: u64,
    royalty_points_numerator: u64,
    presale_mint_time: u64,
    public_sale_mint_price: u64,
    presale_mint_price: u64,
    public_sale_mint_time: u64,
)acquires ResourceInfo,CandyMachine{
    let account_addr = signer::address_of(account);
    let resource_data = borrow_global<ResourceInfo>(candymachine);
    let now = aptos_framework::timestamp::now_seconds();
    assert!(public_sale_mint_time >=  now && presale_mint_time >= now, EINVALI
D_MINT_TIME);
    ...
    candy_data.presale_mint_time = presale_mint_time;
```

```
    candy_data.public_sale_mint_time = public_sale_mint_time;
    ...
}
```

6.5 Redundant conditional statement

Severity: Minor

Status: Fixed

Descriptions: Whether to enter the `if (nfts < 1024)` statement in the function `candymachine::create_bit_mask` has no effect on the values of `full_buckets` and `remaining`.

Code Location: sources/candymachine.move, lines 398–402.

candymachine.move

```
fun create_bit_mask(nfts: u64): vector<BitVector>
{
    let full_buckets = nfts/1024;
    let remaining =nfts-full_buckets*1024;
    if (nfts < 1024)
    {
        full_buckets=0;
        remaining= nfts;
    };
    ......
}
```

Suggestion: Delete the `if (nfts < 1024)` statement.

Appendix 1 – Files in Scope

The following are the SHA1 hashes of the last reviewed files.

Files	SHA–1 Hash
sources/bucket_table.move	c7a814a44b2c95fe7f711e13ae8269a796add086

sources/candymachine.move	a5d80d942cbb191c300cbb02b2731474e38fdbfb
sources/merkle_proof.move	f471046c5245d19a97ee2960af7ae9a453c01100

Appendix 2 – Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as–is, where–is, and as–available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.

MOVEBIT

Securing the Move Ecosystem

https://twitter.com/movebit_

contact@movebit.xyz