MOVEBIT

Securing the Move Ecosystem

# MoveDID Aptos Contracts Audit Report

https://twitter.com/movebit_

contact@movebit.xyz

# MoveDID Aptos Contracts Audit Report

# 1 Executive Summary

## 1.1 Project Information

| | |
|---|---|
| **Type** | DID |
| **Auditors** | MoveBit |
| **Timeline** | 2022-11-17 to 2022-11-29 |
| **Languages** | Move |
| **Methods** | Architecture Review, Unit Testing, Formal Verification, Manual Review |
| **Source Code** | Repository: https://github.com/NonceGeek/MoveDID <https://github.com/NonceGeek/MoveDID> Received Commit: c6488923f5fff6d8ba3b543d319c4a80df449e48 Last Reviewed Commit: 80a3374ccef22cbfeefb4f771880fa8c5266dbf6 |

## 1.2 Issue Statistic

| Item | Count | Fixed | Pending |
|---|---|---|---|
| **Total** | 4 | 4 | |
| **Minor** | 3 | 3 | |
| **Medium** | | | |
| **Major** | 1 | 1 | |
| **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.

# 2 Summary of Findings

MoveDID is a DID protocol that is compatible with Move-based blockchain networks, including Aptos, Sui, and Starcoin. The vision of MoveDID is to be the foundation for the next generation of large-scale Web3 finance and Web3 society. MoveDID could be used for 3 types of subjects: human, organization, and bot. The implementation follows the did@w3c <https://www.w3.org/TR/did-core/> . Our team read the design documents and reviewed the code of the DID project. Our team mainly focused on reviewing the Code Security and normative, then conducted code running tests and business logic security tests on the test net, and performed formal verification with the Move Prover during the test. Our team has been in close contact with the developing team for the past few days. As a result, our team found a total of 4 issues. The teams have discussed these issues together, and the development team has addressed these issues.

# 3 MoveBit Audit BreakDown

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

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- 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

# 4 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 in the conventions in the "Audit Objective", and that can expand to the context 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, and they should actively cooperate (which may include 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 time.

# 5 Findings

## 5.1 Missing max_id update in batch_add_addr

**Severity**: Major

**Status**: Fixed

**Descriptions**: `batch_add_addr` does not modify the `max_id` of the address aggregator. But the `add_addr` function modifies max_id when adding the address. The `max_id` should be increased after `add_addr` and `batch_add_addr`. We found this problem by Prover Formal Verification, refer to chapter 6 for details.

**Code Location**: did-aptos/sources/addr_aggregator.move, line 152,153

addr_aggregator.move

```
    public entry fun batch_add_addr(
        acct: &signer,
        addrs: vector<String>,
        addr_infos : vector<AddrInfo>
    ) acquires AddrAggregator {

        let addrs_length = vector::length(&addrs);
        let addr_infos_length = vector::length(&addr_infos);
        assert!(addrs_length == addr_infos_length, ERR_ADDR_PARAM_VECTOR_LENGHT_MISMATC
H);

        let addr_aggr = borrow_global_mut<AddrAggregator>(signer::address_of(acct));

        let i = 0;
        while (i < addrs_length) {
            let name = vector::borrow<String>(&addrs, i);
            let endpoint = vector::borrow<AddrInfo>(&addr_infos, i);

            table::add(&mut addr_aggr.addr_infos_map, *name, *endpoint);
            vector::push_back(&mut addr_aggr.addrs, *name);

            i = i + 1;
        };
    }
```

**Suggestion**: Increase the `max_id` by 1 inside the while loop.

addr_aggregator.move

```move
// Batch add addr.
public entry fun batch_add_addr(
    acct: &signer,
    addrs: vector<String>,
    addr_infos: vector<AddrInfo>
) acquires AddrAggregator {
    ......
    let i = 0;
    while (i < addrs_length) {
        ......

        table::add(&mut addr_aggr.addr_infos_map, *name, *addr_info);
        vector::push_back(&mut addr_aggr.addrs, *name);

        addr_aggr.max_id = addr_aggr.max_id + 1;

        ......

        i = i + 1;
    };
}
```

## 5.2 Missing semicolon at end of the function

**Severity**: Minor

**Status**: Fixed

**Descriptions**: `;` is missing at the end of the `set_sign_and_updated_at` function. This function does not return a value, adding `;` is more readable.

**Code Location**: did-aptos/sources/addr_info.move, line 114

addr_info.move

```move
public fun set_sign_and_updated_at(addr_info: &mut AddrInfo, sig: vector<u8>, updated_at: u64){
    addr_info.signature = sig;
    addr_info.updated_at = updated_at
}
```

**Suggestion**: Add `;` at the end.

## 5.3 Unused function

**Severity**: Minor

**Status**: Fixed

**Descriptions**: This function is not an entry function and has not been called. Add a comment if you plan to use this afterward, otherwise, delete it.

**Code Location**: did-aptos/sources/addr_info.move, line 118

addr_info.move

```
public fun set_chains_and_description(addr_info: &mut AddrInfo, sig: vector<u8>, updated_at: u64, chains: vector<String>, description: String) {
    addr_info.signature = sig;
    addr_info.updated_at = updated_at;
    addr_info.chains = chains;
    addr_info.description = description;
}
```

**Suggestion**: Remove this or add a comment explaining.

## 5.4 All comments starting with the first letter without capitalization

**Severity**: Minor
**Status**: Fixed

**Descriptions**: The codes have inconsistent comment style with `Aptos Framework` .

**Code Location**: All comments.

**Suggestion**: The first letter of the comment is capitalized, and the end is added with `.` . There must be a space between comments and `//` .

# 6 Prover Formal Verification

The formal verification report of all files and modules is as follows. When using prover verification, you must modify `my_addr` to a valid address.

## addr_aggregator

**General Descriptions**
This module is located in `did-aptos/sources/addr_aggregator.move` .

This module provides related functions to the addition, deletion, and modification of address aggregators.

**Formally Verified Properties**

- Ensured that the `AddrAggregator` resource does not exist under the signer before creating it.

- Verified the postcondition after adding the address to the address aggregator.
- Checked if the address `'0x'` prefix when updating the address info.
- The number of `'address'` is  the same as the number of `'AddrInfo'` when batch-adding addresses to the address aggregators.
- The value of `max_id` should increase 1 after `add_addr` .
- The value of `max_id` should increase the number of addresses after `batch_add_addr` .

**Issue Found**

The `max_id` is not updated in the function `batch_add_addr` .  When judging whether the pre-value of `max_id` plus the number of addresses is equal to the post-value of `max_id` after `batch_add_addr` with the Prover Formal Verification,  the Move Prover successfully verified `add_addr` , but reported an error for `batch_add_addr` .

The `add_addr`  function Prover Formal Verification:

The code `ensures addr_aggr_post.max_id == addr_aggr.max_id + 1;` verified the value of `max_id` should increase 1 after `add_addr` .

addr_aggregator.move

```
    public entry fun add_addr(acct: &signer,
                             addr_type: u64,
                             addr: String,
                             pubkey: String,
                             chains: vector<String>,
                             description: String) acquires AddrAggregator {
        // check addr is 0x begin
        addr_info::check_addr_prefix(addr);

        let addr_aggr = borrow_global_mut<AddrAggregator>(signer::address_of(acct));

        //check addr is already exist
        assert!(!exist_addr_by_map(&mut addr_aggr.addr_infos_map, addr), ERR_ADDR_ALREA
DY_EXSIT);

        let id = addr_aggr.max_id + 1;
        let addr_info = addr_info::init_addr_info(id, addr_type, addr, pubkey, &chains,
 description);
        table::add(&mut addr_aggr.addr_infos_map, addr, addr_info);
        vector::push_back(&mut addr_aggr.addrs, addr);

        addr_aggr.max_id = addr_aggr.max_id + 1;
    }

    spec add_addr{
        include addr_info::CheckAddrPrefix;
        let account = signer::address_of(acct);
        let addr_aggr = global<AddrAggregator>(account);
        let post addr_aggr_post = global<AddrAggregator>(account);
        ensures !spec_exist_addr_by_map(addr_aggr.addr_infos_map, addr);
        ensures spec_exist_addr_by_map(addr_aggr_post.addr_infos_map,addr);
        ensures contains(addr_aggr_post.addrs, addr);
        ensures addr_aggr.max_id + 1 <= MAX_U64;
        ensures addr_aggr_post.max_id == addr_aggr.max_id + 1;
    }
```

The verification result is as follows:

addr_aggregator.move

```
$ did-aptos % aptos move prove --filter addr_aggregator.move
[INFO] preparing module 0xd315808be49fad030c410a317fca3a90ed12106d4ce5e0393b7a874c1a1f2
0e0::addr_aggregator
[INFO] transforming bytecode
[INFO] generating verification conditions
[INFO] 10 verification conditions
[INFO] running solver
[INFO] 0.607s build, 0.447s trafo, 0.027s gen, 2.392s verify, total 3.474s
{
  "Result": "Success"
}
```

The `batch_add_addr` function Prover Formal Verification:

The code `ensures addr_aggr_post.max_id == addr_aggr.max_id + addrs_length;` verified the value of `max_id` should increase the number of addresses after `batch_add_addr` .

addr_aggregator.move

```
    public entry fun batch_add_addr(
        acct: &signer,
        addrs: vector<String>,
        addr_infos: vector<AddrInfo>
    ) acquires AddrAggregator {
        let addrs_length = vector::length(&addrs);
        let addr_infos_length = vector::length(&addr_infos);
        assert!(addrs_length == addr_infos_length, ERR_ADDR_PARAM_VECTOR_LENGHT_MISMATCH);

        let addr_aggr = borrow_global_mut<AddrAggregator>(signer::address_of(acct));

        let i = 0;
        while (i < addrs_length) {
            let name = vector::borrow<String>(&addrs, i);
            let addr_info = vector::borrow<AddrInfo>(&addr_infos, i);

            table::add(&mut addr_aggr.addr_infos_map, *name, *addr_info);
            vector::push_back(&mut addr_aggr.addrs, *name);
            i = i + 1;
        };
    }

    spec batch_add_addr(
        acct: &signer,
        addrs: vector<String>,
        addr_infos : vector<AddrInfo>
    ) {
        let addrs_length = len(addrs);
        let addr_aggr = global<AddrAggregator>(signer::address_of(acct));
        let post addr_aggr_post = global<AddrAggregator>(signer::address_of(acct));
        ensures len(addrs) == len(addr_infos);
        ensures addr_aggr_post.max_id == addr_aggr.max_id + addrs_length;

    }
```

The verification result is as follows:

There is an error message pointing to the validation of `max_id` in Prover: `post-condition does not hold` . So we reviewed the code of the `bacth_add_addr` function, and we found the value of `max_id` does not increase after `batch_add_addr` .

addr_aggregator.move

```
$ did-aptos % aptos move prove --filter addr_aggregator.move
[INFO] preparing module 0xd315808be49fad030c410a317fca3a90ed12106d4ce5e0393b7a874c1a1f2
0e0::addr_aggregator
[INFO] transforming bytecode
[INFO] generating verification conditions
[INFO] 10 verification conditions
[INFO] running solver
[INFO] 0.445s build, 0.420s trafo, 0.018s gen, 2.441s verify, total 3.324s
error: post-condition does not hold
    ┌─ /Users/edy/MoveDID/did-aptos/sources/addr_aggregator.spec.move:53:9
    │
53  │         ensures addr_aggr_post.max_id != addr_aggr.max_id;
    │         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    │
```

We suggest enclosing the push and `max_id` update in the same function so that programmers will never forget to do so. This can be formulated as a struct invariant.

## addr_aptos

### General Descriptions

This module is located in `did-aptos/sources/addr_aptos.move.`

This module provides a function to update the signature and the update time for the Aptos type address.

### Formally Verified Properties

- Verified the abort condition of the `'update_addr'` function.
- Make sure the type of `'AddrInfo'` under the caller is `'ADDR_TYPE_APTOS'` .

## addr_eth

### General Descriptions

This module is located in `did-aptos/sources/addr_eth.move.`

This module provides a function to update the signature and the update time for the Eth type address.

### Formally Verified Properties

- Verified the abort condition of the `'update_addr'` function.
- Make sure the type of `'AddrInfo'` under the caller is `'ADDR_TYPE_ETH'` .

## addr_info

### General Descriptions

This module is located in `did-aptos/sources/addr_info.move.`
This module provides related functions to modify and read address info under the aggregators.

**Formally Verified Properties**

- Verified the abort condition of the `'check_addr_prefix'` function.
- The signature must be verified when updating the msg of address info.

## endpoint_aggregator

**General Descriptions**
This module is located in `did-aptos/sources/endpoint_aggregator.move.`

This module provides functions to operate the endpoint aggregator.

**Formally Verified Properties**

- This resource does not exist in `acct` when creating an endpoint aggregator.
- When operating the endpoint aggregator, `EndpointAggregator` must exist in `acct`.
- After related operations on `EndpointAggregator`, the value of `EndpointAggregator` is as expected.

## eth_sig_verifier

**General Descriptions**
This module is located in `did-aptos/sources/eth_sig_verifier.move.`

Use the public key to verify the signature of eth.

**Formally Verified Properties**

- Added validation of loop invariants for `pubkey_to_address`.
- Covers some `aborts if` cases.

## init

**General Descriptions**

This module is located in `did-aptos/sources/init.move.`

Combine addr_aggregator and endpoint_aggregator init

**Formally Verified Properties**

Verified. similar to addr_aggregator and endpoint_aggregator.

## utils

**General Descriptions**

This module is located in `did-aptos/sources/utils.move.`

This module is mainly some tool class functions.

**Formally Verified Properties**

- Added validation of loop invariants.

# Appendix 1 - Files in Scope

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

| Files | SHA-1 Hash |
|---|---|
| did-aptos/sources/addr_aggregator.move | 868b1315d62920451917081524469cd4dc18af89 |
| did-aptos/sources/addr_aptos.move | 0a2a744a91472d18554dc1f67099c5896ed2458d |
| did-aptos/sources/addr_eth.move | b1cd43ee903ad39c6e09201f544ae4131c1e52b5 |
| did-aptos/sources/addr_info.move | b9f522eab33b2842f133c8ba2803184accfdc985 |
| did-aptos/sources/endpoint_aggregator.move | b234b14a841c56d8fd1f83286c1631e2dc132f88 |
| did-aptos/sources/eth_sig_verifier.move | f10ceef9e0675825be8fe3494891453105f3c0df |
| did-aptos/sources/init.move | 7be11c235834fce708c18684fff6959b16bc8830 |
| did-aptos/sources/utils.move | 884e40fffd48779470559b2af6c28941f3cda03e |
| did-aptos/Move.toml | e7bb209a3851a5dd3af9066c46dc2824e3b9e2aa |

# Appendix 2 - Disclaimer

MOVEBIT

Securing the Move Ecosystem

https://twitter.com/movebit_

contact@movebit.xyz