Beosin launched the Move Lint static detection tool to improve the security of Sui smart contract development through best practices

Beosin · Follow
8 min read · 1 day ago

Listen Share

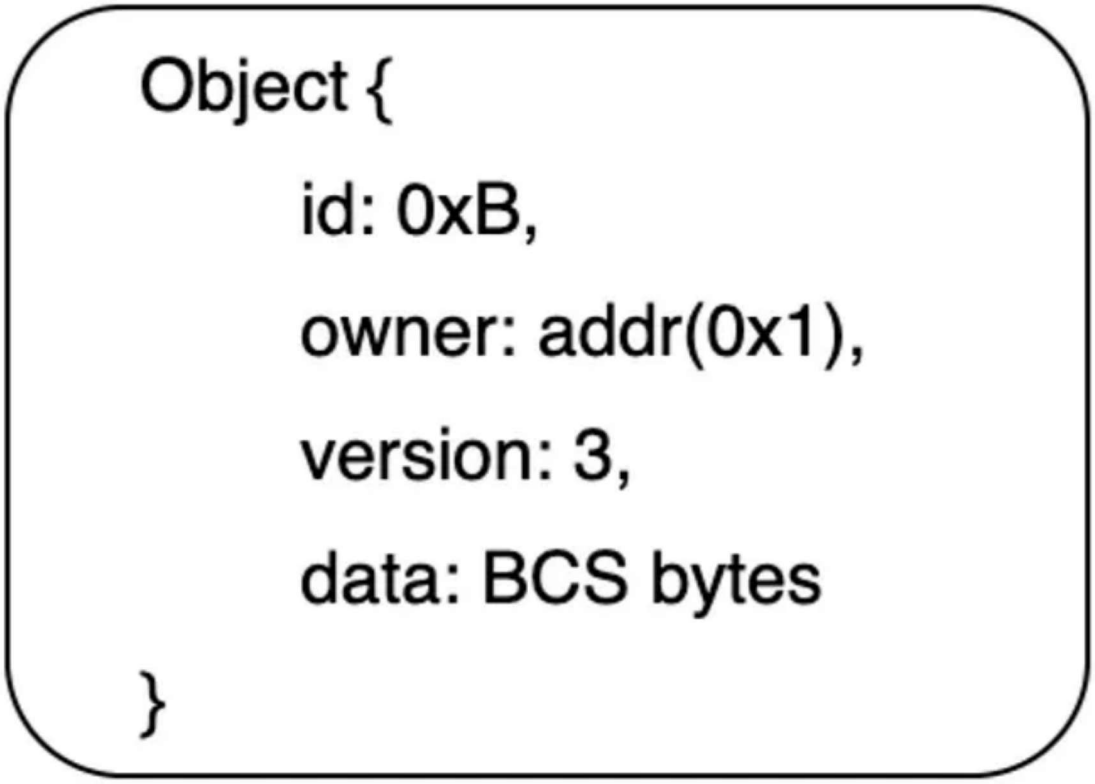
On the evening of May 3, the main network of Sui, a Move public blockchain developed by Mysten Labs, was successfully launched. As a new public blockchain, Sui uses an object-centric data model to facilitate object-oriented programming for developers and provides developers with the Sui Move programming language and the Move Prover detection tool to develop safe and reliable dApps.

If you want to do smart contract development on Sui, what should you pay attention to? Beosin recently launched the Move Lint static detection tool to improve the security of Sui smart contract development through best practices.

1. What is Sui?

Sui is a high-performance public blockchain created by Mysten Labs, enabling developers to build low-latency, high-throughput applications on Sui. Mysten Labs was founded by Evan Cheng, the former head of Facebook's Novi project. It raised $36 million in December 2021 and $300 million in September 2022 at a valuation of more than $2 billion.

Sui is characterized by an object-centric data model. Each object will store a global, unique ID, a copy of the owner's metadata, a version number (version: it will increase by 1 each time the object is called) and binary canonical serialization data (Binary Canonical Serialization). As the figure shown below:

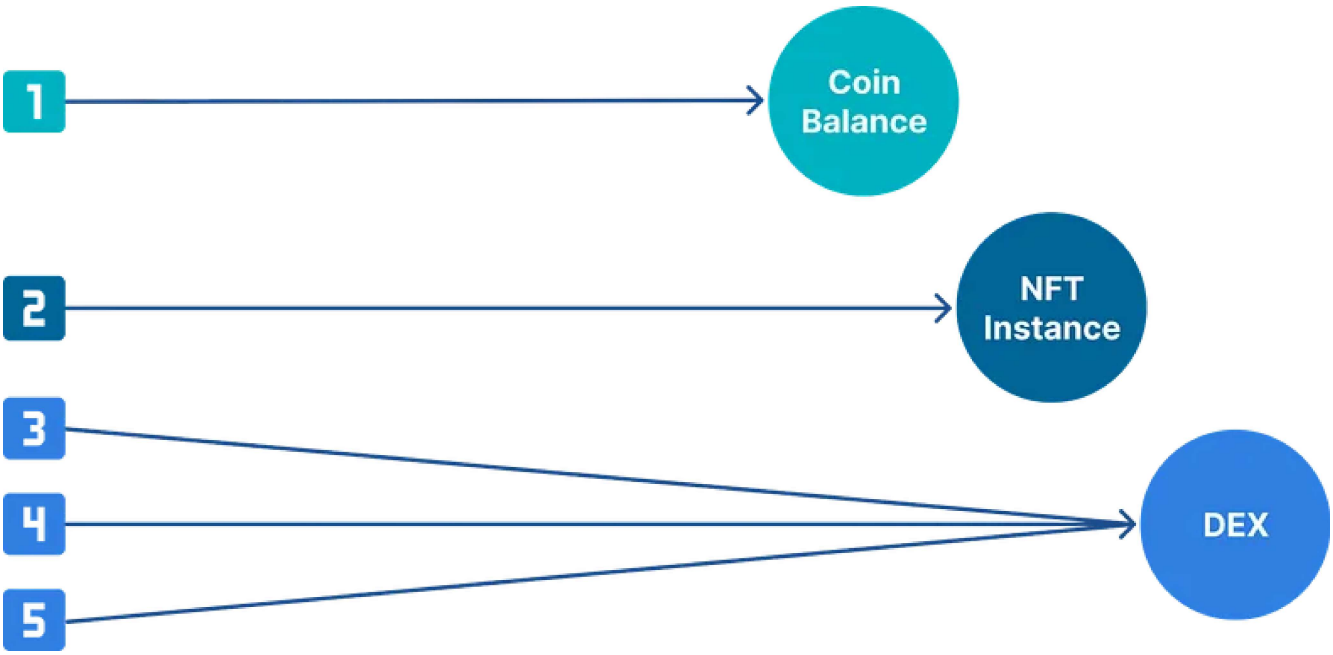


Due to the adoption of the object data model, Sui can group transactions according to whether the objects in different transactions depend on each other, so that different transactions can be processed at different nodes, and multiple transactions can be processed in parallel.

Sui divides objects into owned object and shared object.

Owned object usage scenarios include tokens and NFTs. For transactions containing only owned objects, Sui uses the Byzantine Consistent Broadcast (BCB) consensus algorithm to confirm transactions. The BCB consensus algorithm can be simply understood as firstly, the verifier votes whether to package the transaction, then the transaction initiator counts the voting results, and then the verifier checks whether the statistical results are correct to decide whether to package the transaction. The advantage of this algorithm is that the statistical process is executed on the client side, which reduces the communication time between validator nodes, thereby quickly confirming transactions.

Shared object usage scenarios are DeFi, NFT trading markets, games and other applications that require frequent interaction with users. For transactions involving shared objects, Sui adopts Narwhal and Bullshark protocols for sorting and verification. Narwhal is Sui's transaction mempool, responsible for examining pending transactions and generating a DAG path to traverse them. Bullshark achieves consensus on a specific DAG traversal, thereby confirming a specific order of these transactions.



Based on the above design, the TPS currently tested by Sui has reached a maximum of 297,000. The transaction confirmation time is about 480ms, which shows excellent performance.

2. Move Language Features

Move is a safe and reliable smart contract programming language designed and developed by the Facebook team for the Diem blockchain project. Although Diem

was abandoned by Facebook, the Move language has attracted the attention of capital and developers because of its design features and security. Below we will introduce the important basic concepts of the Move language and the differences between Sui Move.

2.1 Structure (struct)

Move uses struct to define structures. Structures can be empty or contain complex data. This is the only way to create custom types in Move.

2.2 Abilities

Each type of Move can be modified by four keywords to clarify resource permissions. These four keywords we call abilities, they are:

- Copy — the decorated value can be copied.
- Drop — the decorated value can be destroyed when the scope ends.
- Key — the decorated value can be used as a key to retrieve the global state.
- Store — the decorated value can be stored in the global state.

2.3 resources

In Move, resources can be viewed as structure + capability. If a structure is modified by drop, store and key, the resource cannot be copied. This prevents the issue of token issuance. In addition, because resources can be customized by developers through struct, all kinds of complex real-world assets have the potential to be programmed as digital assets.

2.4 Modules

In Move development, code is organized in the form of modules. Modules in Move are similar to smart contracts in other blockchains. Developers declare resource types and methods in modules that define the rules for creating, destroying, and updating declared resources.

However, Sui adopts an object-based data model, and an object can be simply understood as a resource + id. Therefore, in the development of Sui Move, only a structure modified by the keyword key and whose first field is a UID type named id can be used as an object:

```
struct Car has key {
id: UID,
```

```
speed: u8,
acceleration: u8,
handling: u8
}
```

Because Sui Move's global storage is keyed by the object's id, Sui Move can transfer any object after confirming that the ownership is correct. Compared with Move's global storage keyed by (address, type name), Sui Move simplifies the complexity of asset transfer.

3. Move security analysis

3.1 Security by design

Move is designed as a statically strongly typed language with many built-in security features to prevent problems such as arithmetic overflow and permission leakage caused by default visibility; it is designed to be resource/object-oriented programming, and resources and permissions are separated, allowing developers Use the four keywords of copy, drop, store, and key to set the permission of the resource; it is designed as a static call, and dynamic calls are prohibited to avoid re-entry attacks.

3.2 Underlying security

Smart contracts developed by Move undergo byte verification before execution. Move has a built-in bytecode verifier to check resource, type and memory safety. This checks for many common errors before executing the contract and protects the contract from malicious code.

3.3 User level security

Because Move uses static calls, wallet service providers can inform users of the contract execution results after pre-executing the contract. At the same time, due to the resource permission design of Move, the wallet service provider can also clearly understand which permissions the contract will obtain from a user. This information can be displayed to users on the wallet page, helping them to clearly know the possible consequences of the transaction before signing the transaction when interacting with dApps, and avoiding many phishing scams.

4. What should be paid attention to when doing smart contract development on Sui

4.1 Developers need to conduct unit tests and integration tests on the modules they write.

Sui Move provides related support for code testing:

● Use [test], [test_only], [expected_failure] for unit testing

● Use sui::test_scenario to simulate a test scenario with multiple transactions and multiple senders.

● Use sui::test_utilsmodule for better error-correcting messages via assert_eq tests, debug printing via print, and test-only destruction via destroy.

● Use sui move test — coverage to calculate code coverage information at test time, use sui move coverage source — module to see uncovered lines highlighted in red.

4.2 Developers need to use Move Prover and Move Specification Language verification program

The Move language provides the Move Prover formal verification tool to verify the contract. Developers need to use Move Specification Language in the module to be tested to set the verification conditions of the contract logic to verify the correctness of the language level and business level, such as checking the split function in the balance.move code of Sui Framework:

```
/// Split a `Balance` and take a sub balance from it.
public fun split<T>(self: &mut Balance<T>, value: u64): Balance<T> {
    assert!(self.value >= value, ENotEnough);
    self.value = self.value - value;
    Balance { value }
}

spec split {
    aborts_if self.value < value with ENotEnough;
    ensures self.value == old(self.value) - value;
    ensures result.value == value;
}
```

4.3 Developers need to pay attention to the security issues of different programming modes

Capability is a programming pattern that allows authorized operations on objects. In this mode, developers need to pay attention to whether the permissions are

correct during the creation of the capability, use and destruction. Best practice is to use capability in the init function to ensure that there is only one capability and it is controlled by the module publisher:

```
struct AdminCapability has key {
 id: UID
}

fun init(ctx: &mut TxContext) {
 transfer::transfer(AdminCapability {
  id: object::new(ctx),
 }, tx_context::sender(ctx))
}
```

Witness is a programming pattern for controlling the one-time creation of objects. If a Witness can be copied or created publicly, then the object it protects is likely to be attacked. Best practice is to use the drop keyword to ensure the Witness is destroyed when the scope ends:

```
/// the types passed here must have `drop`.
    struct Guardian<phantom T: drop> has key, store {
        id: UID
    }
/// The first argument of this function is an actual instance of the type T wi
    public fun create_guardian<T: drop>(
        _witness: T, ctx: &mut TxContext
    ): Guardian<T> {
        Guardian { id: object::new(ctx) }
    }
```

In addition, developers also need to pay attention to whether the third-party modules relied on during development have been deprecated or updated. If the dependent code library is inconsistent with the code deployed on the chain, there may be potential risks. Some third-party modules may even contain malicious code.

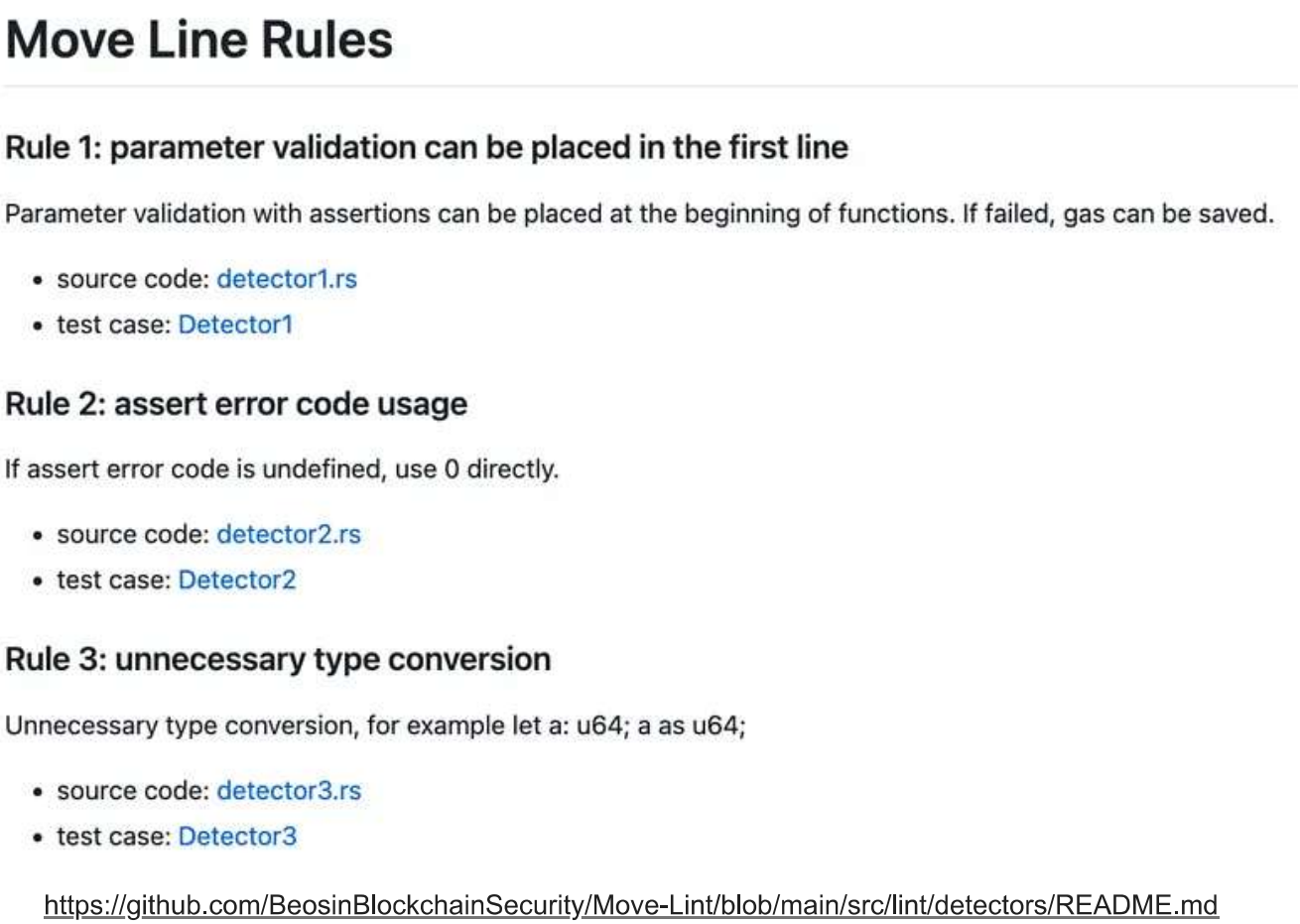
Currently, Beosin has launched the Move Lint static detection tool, which is a special version of Beosin VaaS. This tool only needs the user to import the smart contract to automatically discover potential security risks in the contract, locate the

location of the vulnerability, and enhance the security of the contract safety. **The tool mainly includes two aspects of detection:**

(1) Code conventions detection. It detects any violations of certain code conventions during the development of a smart contract. The possibility of introducing security vulnerabilities is greatly increased by inappropriate code writing.

(2) Regular security issues detection. It detects regular security issues in the contracts. Regular security issues are issues that may exist in any contract and are independent of business logic, e.g. division before multiplication.

The tool comes with a detection rule library and provides a convenient interface for adding and deleting detection rules. With the update of the developer's Move contract and business model, developers can quickly adapt to new detection rules and improve the detection capabilities of tools. Currently, there are 8 detection rules in the detection rule base, covering third-party library version and deprecation detection, assertion writing detection and general code error detection:



Move Line Rules

Rule 1: parameter validation can be placed in the first line

Parameter validation with assertions can be placed at the beginning of functions. If failed, gas can be saved.

- source code: detector1.rs
- test case: Detector1

Rule 2: assert error code usage

If assert error code is undefined, use 0 directly.

- source code: detector2.rs
- test case: Detector2

Rule 3: unnecessary type conversion

Unnecessary type conversion, for example let a: u64; a as u64;

- source code: detector3.rs
- test case: Detector3

https://github.com/BeosinBlockchainSecurity/Move-Lint/blob/main/src/lint/detectors/README.md

Developers are welcome to use Move Lint in development and customize more detection rules. The tool links are as

follows: https://github.com/BeosinBlockchainSecurity/Move-Lint.

If you have any feedback, welcome to raise an issue on GitHub or contact us by email.