SharkTeam

SharkTeam: Contract Development and Contract Security for Move

Oct 13 2022

SharkTeam, a leading blockchain security service team, offers smart contract audit services for developers. To satisfy the demands of different clients, thesmart contract audit services provide both manual auditing and automated auditing.

We implement almost 200 auditing contents that cover four aspects: high-level language layer, virtual machine layer, blockchain layer, and business logiclayer, ensuring that smart contracts are completely guaranteed and Safe.

Recently, around Aptos and Sui, the emerging high-performance L1 chain and the Move smart contract programming language behind these new chains have attracted a lot of attention, the community is also very active, and many developers and projects have begun to actively switch to Move. But Move is quite different from Solidity, and even Rust, which is relatively close, has many differences. So, how can we use Move to develop excellent Dapps, and how can we ensure the security of contracts and businesses? This is a crucial question.

SharkTeam will have an in-depth discussion on Move contract development and contract security. This is the first part where we will focus on the development of the Move contract.

1.Background

Move is a new programming language, a secure and reliable smart contract language developed by Meta (formerly Facebook) for its Diem project.

Meta (formerly Facebook) released the global stable digital currency project Libra in 2019, and designed and developed a safe and reliable smart contract language Move for it.

Libra suddenly announced in December 2020 that it would change its name to Diem and confirmed that it will be capped in early 2021. At the same time, the original Libra Association was renamed the Diem Association. Subsequently, it was forced to transform due to regulatory restrictions, and finally, in early January 2022, it chose to sell, Diem died without success, and its cryptocurrency dream was shattered.

Although the Diem project has not brought great opportunities and changes to the industry, its legacy, the Move development language, has attracted the attention of project parties and developers. The reason is that the Move language is more suitable for the development of digital assets than Solidity, and has higher security, making up for the loopholes and defects in Solidity and EVM to a certain extent. The public chain ecology derived from the Move language is in the ascendant, such as Aptos, Sui, Linera, Starcoin, etc.

2.Language features of Move

We summarize the 6 major features of the Move language as follows:

(1) First-class resources and digital assets

(2) Authority separation and operability

(3) Module and flexible combination

(4) Generics and programming flexibility

(5) Formal verification mechanism

(6) Static call and virtual machine sandbox

2.1 First-class resources and digital assets

The Move language is a resource-oriented language, and its resource-oriented concept is where the Move language gets the most attention. Move can customize resources, resources have programmability and security. Developers can customize resources through programming, and truly realize unique digital assets.

Compared with the asset defined in Solidity, which is just a numerical value, the asset does not have unique characteristics, nor does it have unique characteristics, that is, only digitalization is realized, but digital assetization is not realized. For example, Solidity issues a Token, deploys a Token contract, and mints a certain number of Tokens for an account A. In fact, it only assigns a certain value to account A in the Token contract. The transfer, such as A transfer to B, is to modify the value of account A and account B in the Token contract, that is, to reduce the value of A and increase the value of B.

After Move creates a resource Token and mints a token for account A, it creates a Token resource and saves it in account A. This resource is unique, and only account A has operation authority, which truly realizes digital assetization, not just digitalization. .

The resources in Move contain 3 features:

(1) Some numerical properties of digital resources are represented in numerical form. For example, the resource is internally represented in numerical form to represent the amount of digital currency, which is stored as a data structure and can be used as parameters and return values, similar to the structure in Solidity. ;

(2) Unique digital assets that cannot be copied, discarded or reused, can be safely stored and transferred, and can only be created and destroyed by the module that defines the resource. This form of resource realizes digital in the true sense. Assets, which are completely different from the numerical values in Solidity that represent digital assets. This makes digital assets more secure in the process of creation, use and destruction, and will not be generated, added or lost out of thin air, and can avoid reentrancy vulnerabilities in Solidity.

(3) The storage and reading of resources are bound to the account. After the resource is created, if it is not destroyed, it must be stored under an account, and must be used after being taken out. Therefore, only when an account is allocated can the corresponding resources be held, and then the resources can be operated. In addition, when reading the resources in the account, you can choose two types: immutable and variable to limit the modification of resources and improve the security of the contract.

2.2 Permission separation and operability

The resource definition and permissions in the Move language are separated. The default permission is only move, and it is clear that the permissions of the resource belong to the user. Compared to Solidity, where data permissions are assigned to contracts, this makes resources more secure.

On the other hand, in order to implement more and more complex services, Move defines four additional permission attributes: copy, drop, store, and key. These four attributes can be combined arbitrarily. In the resource definition, the permission attribute of the resource can be clearly specified, which is convenient for users to define resources with various permissions, and brings users greater and more flexible operability. for example:

• Use the combination of copy + drop + store + key to define the most common resource, similar to the structure in Solidity;

• Using the combination of drop + store + key, the defined resources cannot be copied, which can avoid the problem of additional copying;

• Using the combination of copy + drop, the defined resources cannot be stored and retrieved, and can be used for temporary copyable common resources;

• Using drop alone, the defined resources can only be moved and discarded, but cannot be copied, stored, or retrieved, and can be used for temporary unique resources;

• Without using any permission type, the defined resource can only be moved.

In different application scenarios, different permission attributes need to be used in combination when defining resources. For security reasons, the permission selection of resources should follow the principle of minimum. For example, if the resource does not have a copy scenario, the permission statement should not include copy.

2.3 Modules and Flexible Combinability

The Move language uses the design of modules and scripts. Among them, the module is similar to the Contract in Solidity, which defines the resource data and related business logic in the contract, and the script defines the transaction.

The Move language itself is a static language, in the form of static calls, and the contract calls are executed in the virtual machine sandbox. During the calling process, the state of the contract is isolated by the security inside the programming language, rather than relying on the virtual machine for isolation, which allows different modules to be combined and called more flexibly in the virtual machine sandbox directly.

When a module is optimized and upgraded, all other contracts that have used the module in combination will automatically call the latest version after the upgrade. Compared with a series of complex operations brought about by contract upgrades in Solidity, the modules in the Move language have strong composability, flexibility, and fast contract optimization and upgrade speed.

In terms of security, modules do not support circular recursive dependencies, which fundamentally avoids the risk of contract reentrancy.

2.4 Generics and programming flexibility

The use of generics greatly increases the flexibility of Move. We can introduce generics for data structures and function methods, so that code with the same function only needs to be implemented once and can be applied to multiple types, similar to C++ generic programming.

2.5 Formal verification mechanism

The Move language has a component called Move Prover (MVP) that supports formal verification mechanisms as an off-chain static security verification tool. MVP verifies that a program conforms to a certain specification through an automatic theorem proving solver in the field of formal verification.

This method of formal verification requires a detailed understanding of the contract operation logic, and the specifications required by the custom program logic, and then communicated to the validator MVP together with the program. Only after the MVP is verified, the contract code can be executed in the runtime environment (VM runtime).

In order to better realize the specification of formal verification, Move has customized the specification prediction Move Specification Language. The normative prophecy defines preconditions, postconditions, termination conditions, demand conditions, determination conditions, modification conditions, constant conditions, assumptions and assertions to realize the verification conditions required for contract logic, including language-level verification requirements and business level verification requirements.

2.6 Static call and virtual machine sandbox

Static call means that when the program is called, the calling object has been determined before running, and the calling object is not changed during the running process.

The Move language uses a static calling system, which is a logical constraint, and can find some low-level errors and vulnerabilities in programming before running. For example, integer overflow vulnerabilities can be found before running, which can avoid attackers in dynamic systems. Attacks triggered by malicious contracts calling project-side functions. Compared with dynamic calls, static calls can avoid the security problems of dynamic calls to a certain extent, improve the security of smart contract operation, and also improve the stability of blockchain transactions.

The Move Virtual Machine (MoveVM) is a stack machine with a static type system for executing transactions represented by Move bytecodes and consists of two main components: the core VM and the VM runtime. Among them, the core VM contains the underlying data of MoveVM, and the VM runtime provides an environment for executing transactions.

Contract execution and invocation are all in the same Move virtual machine sandbox. During this process, the states of different contracts are mainly isolated by the security inside the program bytecode. The key is to determine the modules and static systems of the Move language. characteristics and the binding relationship between Move resources and accounts.

First of all, the definition of a module is static, deterministic, and somewhat closed. A function in a module can only modify the resources defined and created by the module. This modification is based on the pre-defined function. To access and modify resources created by other module definitions, you need to call functions of other modules, that is, static calls.

Compared with Solidity's dynamic call, the function is called by address and signature, and the called object (such as the call function) can only be determined when it is actually executed. The determinism of static call makes the execution of the Move function more secure.

Resources are bound to accounts, and account authorization is required to modify resource data. When executing a contract, you can only modify the resources of the related account and have no effect on other accounts. This limits the scope of the contract's influence. Even if the contract has logical loopholes, it cannot affect the resources of all accounts at one time. In Solidity, asset data is stored in the contract. If there is a loophole in the contract, all assets will be affected. In comparison, the Move contract is more secure.

3 Move & Solidity Security Analysis

Move is a secure resource-oriented static language. Compared with Solidity, we summarize the following points from a security perspective:

(1) Asset security

•Move defines assets in the form of resources, which are unique and cannot be copied, lost or reused, ensuring the security of assets and avoiding risks such as unlimited additional issuance, asset reuse, and asset loss;

•Solidity defines assets in numerical form, which can be copied and reused, which makes assets subject to certain risks, such as re-entry attacks, assets being reused; unlimited asset issuance, asset loss, etc.

(2) Permission security

•Resource definitions and permissions in Move are separated, and four permission attributes of copy, drop, store, and key can be used flexibly to limit the operation of resources, which ensures the security of permissions of assets. Risk due to permissions. Therefore, care needs to be taken when defining permissions.

• In Solidiy, since assets are numerical values, there is no finer-grained permission distinction, that is, any asset has all permissions.

(3) Verify security

• In addition to the compiler checking syntax issues, the Move verifier MVP and custom specification language are more important in Move. MVP verifies whether the transaction bytecode meets the conditions defined by the specification through formal verification. Since the normative conditions are user-defined, it is necessary to be familiar with detailed business processes when defining them, so as to avoid loopholes in the specification as much as possible, especially some normative conditions at the business level.

• There are no validators in Solidity, so it is more prone to some low-level security vulnerabilities, such as state variable initialization vulnerabilities.

(4) Static security

•Move is a static language. Function calls use deterministic static calls and do not support cyclic recursive dependencies, which can avoid some security problems in dynamic calls, such as reentrancy vulnerabilities, risks brought by the same function signature, and delegatecall. Risks (such as state variable order conflicts in proxy mode).

•Solidity is a dynamic language, and naturally there will be some security problems caused by dynamic calls, such as the built-in call function is more prone to reentrancy vulnerabilities, and the vulnerabilities caused by state variable conflicts in the proxy upgrade mode.

(5) Storage security

•Assets in Move are stored under accounts, and only accounts can operate. Even if there are logical loopholes in the module defining assets, it will not threaten the assets of all accounts, only a few assets will be affected, which improves the security of asset storage .

•Assets in Solidity are stored in the contract account in the form of numerical values. When there are loopholes in the contract, all users' assets will be at security risk.

To sum up, Move is a language with a high security level and basically avoids most language-level vulnerabilities. However, there is still a strong audit requirement for the Move contract. The reason is that although Move has customized the specification language and can customize the business verification, it needs to be very familiar with the business. Therefore, if the business complexity is high, or the developer is not familiar with the business, there may be logical loopholes in the business specifications in the contract, threatening the security of the project and user assets.

4 Summary

Through the above analysis, Move is a secure resource-oriented static smart contract language. Some designs of the language itself, as well as formal verification tools, improve the security of Move language level. Therefore, the project can avoid many low-level vulnerabilities during the development process, especially the language-level vulnerabilities, such as reentrancy, integer overflow, variable initialization vulnerabilities, etc. These are things that Solidity doesn't have. This is also an important reason why Aptos, Sui and other public chains choose Move.

However, the public chain derived from Move is still in the development stage, and there are not many projects that are actually running. It will take time to verify whether Move can really stand the test.

In the next lesson, we'll focus on security auditing points for the Move project.

About Us

Our vision is to improve security globally. We believe that by building this security barrier, we can significantly improve lives around the world.SharkTeam composes of members with many years of cyber security experiences and blockchain, team members are based in Suzhou, Beijing, Nanjing and Silicon Valley, proficient in the underlying theories of blockchain and smart contracts, and we provide comprehensive services including threat modeling, smart contract auditing, emergency response, etc. SharkTeam has established strategic and long-term cooperations with key players in many areas of the blockchain ecosystem, such as Huobi Global, OKX, polygon, Polkadot, imToken, ChainIDE, etc

SharkTeam

In Math, We Trust!