Potential Risk Detection System of Hyperledger Fabric Smart Contract based on Static Analysis

Penghui Lv, Yu Wang, Yazhe Wang, Han Wang and Qihui Zhou

March 16, 2021

Potential Risk Detection System of Hyperledger Fabric Smart Contract based on Static Analysis

Penghui Lv*†, Yu Wang*, YaZhe Wang*, Han Wang*†, Qihui Zhou*†

**Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China*

† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

*{lvpenghui, wangyu, wangyazhe, wanghan, zhouqihui}@iie.ac.cn

Abstract— The smart contracts of Hyperledger Fabric blockchain are mostly developed in general-purpose programming languages, which are well-known by potential developers, such as Golang. Due to the lack of mature development specifications for smart contracts using general-purpose programming language, there are often potential risks in the smart contracts related to the characteristics of Hyperledger Fabric. It will bring many inconveniences and potential safety hazards to users after the smart contracts are deployed. Although there are already some potential risk detection tools for smart contracts of Hyperledger Fabric, the accuracy and coverage of the tools are limited. In response to the above problems, this article summarizes three types of potential risks in the smart contracts of Hyperledger Fabric: Non-determinism Risk, Logical Security Risk, and Private Data Security Risk. In order to detect these different types of potential risks, we propose a new static analysis method based on Abstract Syntax Tree, Package Dependency Analysis, and Functional Dependency Analysis. At the same time, we design a detection system that can accurately locate the location of potential risk items in the smart contracts of Hyperledger Fabric and generate development suggestions for the reference of smart contract developers.

Index Terms—Hyperledger Fabric, Static Analysis, Smart Contract, Potential Risk Detection

I. INTRODUCTION

The smart contract of Hyperledger Fabric, the most famous permissioned blockchain [1], is called “chaincode”, which is a piece of computer code running on the docker of Hyperledger Fabric. The chaincodes are mostly developed in general-purpose programming languages, e.g., Golang, Java, and NodeJs. Among them, Golang has become the most mainstream development language [2-3]. However, the chaincodes developed by general-purpose programming languages lack mature development specifications, so most developers tend to blur the boundaries between the chaincodes and ordinary applications, and introduce potential risks into the chaincodes during the development process. It may bring many inconveniences and potential safety hazards to users during these chaincodes operation. In addition, once the chaincodes are deployed and executed, it is more difficult to be updated and deleted. Therefore, it is extremely important to discover potential risks in the chaincodes and improve their quality before the chaincodes are deployed.

At present, there is related research on potential risk detection of the chaincodes developed by general-purpose programming languages. The general-purpose programming languages have their own detection tools, such as Gosec[4] and staticCheck[5], which are usually used to identify grammatical errors in the Golang language programming process, but they cannot effectively identify the potential risks associated with the characteristics of smart contract. Zhang et al. [6] summarizes the non-deterministic risks in the chaincodes of Hyperledger Fabric. Huang Y et al. [7] introduced Chaincode scanner, which can only detect 9 risk items related to the non-determinism risks and logical security risks in the chaincodes. Yamashita K et al. [8] designed a static detection tool based on the Abstract Syntax Tree, this tool can detect 14 risks items mainly related to the non-determinism risk and logical security risk in the chaincodes.

Although some papers have summarized some risk items related to chaincode, the introduction of these risk items is not comprehensive enough. In addition, Current tools detect potential risks in the chaincodes based on the Abstract Syntax Tree, the accuracy and coverage of these tools are limited. Particularly, they have shortcomings in detecting potential risks items related to the private data security, these items are also often ignored by developers, and most users are concerned.

In response to the above problems, our article makes the following contributions:

1) This article more comprehensively summarizes the three types of potential risks in the chaincodes, including Non-determinism Risk, Logical Security Risk, and Private Data Security Risk.

2) For detecting different risks of Hyperledger Fabric smart contract, a new static analysis method is proposed, which is based on Abstract Syntax Tree, Package Dependency Analysis, and Functional Dependency Analysis.

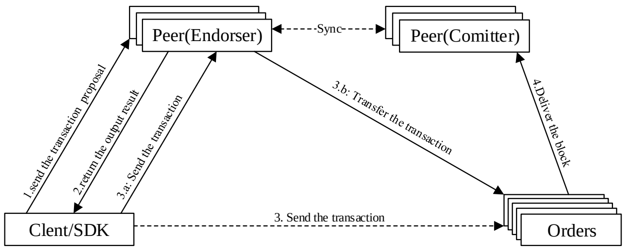
3) A potential risks detection system of the chaincodes is designed based on the new static analysis method. it can detect 16 risk items with high accuracy, of which the Data Privacy Security Risks are more comprehensively detected, and just makes up for the deficiencies of other tools.

Organization: This article is divided into 6 sections. Section II introduces transaction flow of Hyperledger Fabric, and summarizes potential risks in the chaincodes, Section III Section IV provides our system’s architecture, and presents the design and implementation of our system. Section V discusses the results and our system. Section VI draws our conclusions.

II. POTENTIAL RISKS IN HYPERLEDGER FABRIC

A. Hyperledger Fabric

To better understand potential risks in the chaincodes of Hyperledger Fabric, we briefly introduce the transaction flow of Hyperledger Fabric version 1.0 or later. There are the following four steps to complete a transaction, requiring the participation of Peers (acting as endorsing peer or comitter peer), Orderers and Clients [9], as shown in Figure 1.



Peer(Endorser)
Sync
Peer(Comitter)
1.send the transaction proposal
2.return the output result
3.a: Send the transaction
3.b: Transfer the transaction
4.Deliver the block
Clent/SDK
3. Send the transaction
Orders

Figure 1 The transaction flow of Hyperledger Fabric

1) Initiate a transaction. The client sends a transaction proposal to the corresponding endorsing peers.

2)Transaction endorsement. Each endorsing peer takes the received transaction proposal inputs as arguments to invoke the chaincodes, simulates transaction execution, and finally sends the transaction results back to the client. Then, the client packages all transaction results received into a transaction request to Orderers via the endorsing peers.

3)Generating new blocks. The Orderers order the received transactions, generate new blocks, and then deliver these blocks to comitter peers (all peers) on the channel.

4)Transaction validation. When receiving the new blocks, each comitter peer validates the transaction, and then appends the new blocks to the chain. Simultaneously, the result of each valid transaction is generated and stored in the current state database.

The above is the general transaction flow of Hyperledger Fabric. However, if there are potential risks in the chaincodes, it will not only cause the transaction to fail and affect the stable operation of the entire system, but also bring challenges to the user's information security. This article analyzed the chaincodes of Hyperladger Fabric, and summarized the three types of potential risks in the chaincodes: Non-determinism Risk, Logical Security Risk and Privacy Data Security Risk.

B. Non-deterministic Risk

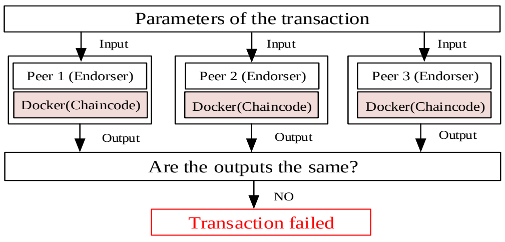


Figure 2 The process of transaction failure caused by non-deterministic risk

In Hyperledger Fabric, transaction failure caused by Non-determinism Risk is reflected in the transaction endorsement step, as shown in Figure 2. Distributed and independent endorsing peers input the same parameters of the transaction, and then execute the chaincodes with Non-deterministic Risk to simulate transactions. But the transaction results from different endorsing peers are not the same, this violates the consensus rules of Hyperledger Fabric and causes the transaction to fail [10]. Hence, it is necessary to find the Non-deterministic Risk in the chaincodes. The Non-deterministic Risk mainly comes from Non-deterministic Data Sources, Non-deterministic Execution Process and Non-deterministic External Calls.

Definition 1: Non-deterministic Data Sources are objects or variables, which may have different values when running in different peers, including Random Number Generation, System Timestamp, and Reified Object Addresses.

Since each endorsing peer stimulates the chaincodes in different environments, it is difficult to ensure that the timestamp functions and random functions of the chaincodes running in different endoring peers have the same result for each peer. Similarly, Reified Object Addresses are addresses of memory, which may be different in different environments.

Listing1: example of Non-deterministic Data Sources

```
// User set a value
setValue := arg[0]
rand.Seed(seed)        //random function
sel := rand.Intn(10)
if setValue == sel { a → b}
else{ a ← b}
```

Listing 1 shows an example of the Non-deterministic Risk from Non-deterministic Data Sources. In this example, if setValue equals sel, User a transfers a sum of funds to user b. However, the random function in the chaincodes running in different endoring peers generates different numbers for each peer, so this transaction will be difficult to succeed.

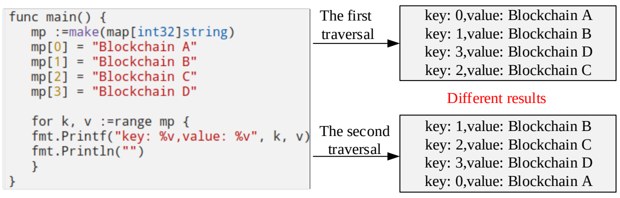
Definition 2: The Non-deterministic Execution Process refers to the process in which the internal logic execution sequence of the same function of the chaincodes in different peers is different, or the values of their same variables become different, which leads to uncertain transaction results.

The factors that cause the Non-deterministic Execution Process are as follows: Global Variable, Field Declarations, Concurrency of Program, Map Structure Iteration.

1) Global Variables and Field Declarations: Due to the differences in the endorsement policy of each peer, not every peer simulates the same transaction, so this may cause the values of the Global Variables and the Field Declarations in each peer to become different.

2) Concurrency of Program: Using Golang language to develop the chaincodes, Concurrency of Program makes the execution order of the chaincodes impossible to determine.

3) Map Structure Iteration: Map Structure Iteration may result in a different sequence of key-value pairs, as shown in Figure 3.



```
func main() {
   mp :=make(map[int32]string)
   mp[0] = "Blockchain A"
   mp[1] = "Blockchain B"
   mp[2] = "Blockchain C"
   mp[3] = "Blockchain D"

   for k, v :=range mp {
   fmt.Printf("key: %v,value: %v", k, v)
   fmt.Println("")
   }
}
```

```
key: 0,value: Blockchain A
key: 1,value: Blockchain B
key: 3,value: Blockchain D
key: 2,value: Blockchain C
```

```
key: 1,value: Blockchain B
key: 2,value: Blockchain C
key: 3,value: Blockchain D
key: 0,value: Blockchain A
```

Figure 3 Non-deterministic results coming from Map Structure Iteration

Definition 3: Non-deterministic External Calls refer to visits from outside the blockchain that lead to uncertain transaction results, including External File Accessing, External Library Calling, Web Service, and System Command Execution.

Non-deterministic External Calls may have different execution logic or uncertain data sources in different peers, so the results obtained by running are not guaranteed to be consistent. Therefore, when encountering Non-deterministic External Calls, developers need to be clear about what results are obtained.

C. Privacy Data Security Risk

Definition 4: Privacy Data Security Risk refers to the risk of transaction failure due to operating authority issues, or sensitive data leakage due to lack of security measures when simulating the chaincodes. It includes Cross Channel Chaincode Invocation, Unencrypted Sensitive Data, and Unused Privacy Data Mechanism.

1) Cross Channel Chaincode Invocation: Hyperledger Fabric provides the function of the Cross Channel Chaincode

Invocation. The chaincodes can invoke other chaincodes on the same channel to access or modify the state database, but cannot call other chaincodes on a different channel to create a new transaction. Therefore, when using cross-channel calling functions, developers should avoid calling the chaincodes on another channel to create a new transaction.

2) Unencrypted Sensitive Data: If there is Unencrypted Sensitive Data in the chaincodes, the plaintext of sensitive transaction data is stored in the ledger, which may cause the leakage of transaction data.

3) Unused Privacy Data Mechanism: For the protection of sensitive data, Hyperledger Fabric also provides a Private Data Mechanism to enhance the security of transaction data [11-12]. If there is Unused Privacy Data Mechanism in the chaincodes, the security of transaction data may be weakened.

D. Logical Security Risk

Logical Security Risk mainly discusses the risk arising from the state database operation of Hyperledger Fabric, such as Range Query Risk and Read Your Write.

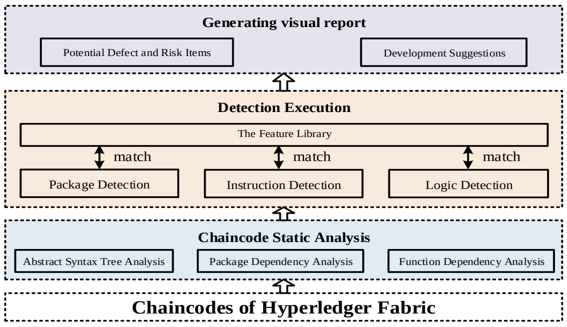
1) Range Query Risk: Hyperledger Fabric provides some range query methods to access the state databases, such as GetQueryResult(), GetPrivateDataQueryResult(), and GetHistoryForKey(). These methods are executed during the endorsement phase, but are not re-executed during the verification phase. Therefore, these methods cannot be used to modify the ledger in the chaincodes, and can only be used to query the transaction of the ledger.

2) Read Your Write: In Hyperledger Fabric, the operation of writing the transaction data of the blockchain into the ledger is executed after the transaction is completed and verified. Therefore, there cannot be a write-and-read operation on a variable of the same process in the chaincodes, that is, Read Your Write are not supported in Hyperledger Fabric.

III. DESIGN AND IMPLEMENTATION OF OUR SYSTEM

A. Architecture of Our System

In order to find out potential defects and risks in the chaincodes of Hyperledger Fabric, we propose a detection technology based on static analysis and implement a detection system for the chaincodes developed by the Golang language, as shown in Figure 4. The system mainly includes three modules: Chaincode Static Analysis, Detection Execution and Generating Visual Report.



Generating visual report
Potential Defect and Risk Items
Development Suggestions
Detection Execution
The Feature Library
match
match
match
Package Detection
Instruction Detection
Logic Detection
Chaincode Static Analysis
Abstract Syntax Tree Analysis
Package Dependency Analysis
Function Dependency Analysis
Chaincodes of Hyperledger Fabric

Figure 4 Architecture of our tool

1)Chaincode Static Analysis. In this module, the chaincodes of Hyperledger Fabric are analyzed to obtain static structure information such as abstract syntax tree, package dependency, and functional dependency.

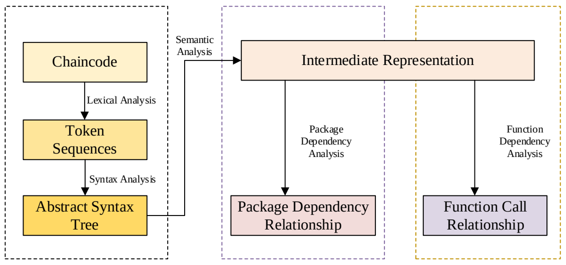
2) Detection Execution. This module is designed to determine the types and locations of potential defects and risks by matching a feature library composed of static features of the risks of chaincodes.

3)Generating Visual Report. The report includes descriptions and locations of potential risk items in the chaincodes, and development suggestions to eliminate these items.

Chaincode Static Analysis and Detection Execution module are the core parts of our detection system. The following mainly introduces the design and implementation of these modules.

B. Chaincode Static Analysis

Chaincode Static Analysis is to perform Abstract Syntax Tree Analysis, Package Dependency Analysis and Function Dependency Analysis on the chaincodes to obtain their static structure information: Abstract Syntax Tree, Package Dependency Relationship, Function Call Relationship, as shown in Figure 5.



Chaincode
Lexical Analysis
Token Sequences
Syntax Analysis
Abstract Syntax Tree
Semantic Analysis
Intermediate Representation
Package Dependency Analysis
Package Dependency Relationship
Function Dependency Analysis
Function Call Relationship

Figure 5 Simplified diagram of Chaincode Static Analysis

C. Abstract Syntax Tree Analysis

Abstract Syntax Tree Analysis is the process of obtaining the Abstract Syntax Tree (AST) of the chaincodes by using Lexical Analysis and Syntax Analysis in the compiler [13].

The chaincodes are used as input to generate recognizable token sequences through Lexical Analysis. Then through Syntax Analysis, the generated Token sequences are rewritten to construct the AST of the chaincodes using the bottom-up analysis method, which is constructed from subtrees, gradually merged upwards, and finally assembled into a complete tree. As shown in Figure 6, the AST of the chaincodes takes the entire file ast.File as the root node, and other nodes describe the grammatical structure of different levels in the file from top to bottom, and each node has its detailed structure declaration and definition, which represents its location in the chaincodes and the relationship with other files.

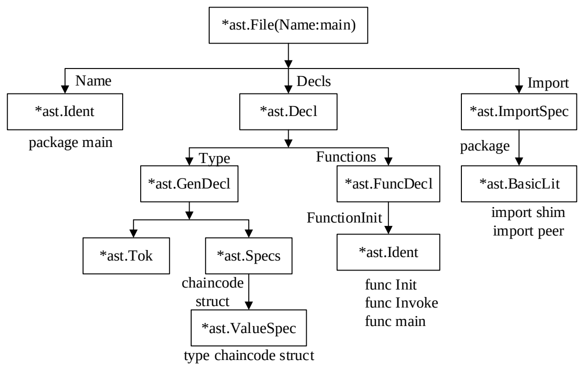


Figure 6 Abstract Syntax Tree of the chaincodes

D. Package Dependency Analysis

Package Dependency Analysis is a process of analyzing the AST of the chaincodes to obtain the dependencies between the packages called by the chaincodes.

Algorithm 1: Package Dependency Analysis

```
1   begin
2       // Convert the AST of the chaincodes into the IR
3       IR := SemanticAnalysis (AST)
4       //Get the parameters of the top-level package
5       root,pkgName,level,imported:= getTopPackage(IR)
```

```
    processPkg(root, pkgName, level, imported ) {
        // Level is not greater than maxLevel
        if level++; level > *maxLevel {return nil;}
        //Get the dependent package list of this layer
        pkg := buildContext.Import(pkgName, root)
        //Get the path of the dependent package
        importPath := normalizeVendor(pkgName)
        pkgs[importPath] = pkg
        // Recursive
        for _, imp := range getImports(pkg) {
            if _, ok := pkgs[imp]; !ok {
            processPkg(pkg.Dir, imp, level, pkgName) }
        }
    }
    return pkgs;
end
```

The algorithm of Package Dependency Analysis is shown in Algorithm 1. This algorithm first uses semantic analysis to convert the AST of the chaincodes into the intermediate representation (IR) with Static Single Assignment (SSA) [14]. Then, the name, path and other parameters of the top-level package are obtained from the IR as the input of the processPkg function, which is used to read the import keyword in the specified package name to get the list of the dependent packages. The algorithm further traverses the list of the dependent packages and calls the processPkg function recursively to get the list of the dependent packages of each layer. Among them, root is the path of the top-level dependent package, pkgName is the name of the top-level dependent package, level is the level of the current package. When the algorithm recursively reads the list of dependent packages from the IR, the standard library provided by Golang language can be read at the third layer at most, so maxLevel is set to 3 [15-16]. Finally, the package dependency relationship of the chaincodes is obtained, as shown in Figure 7.

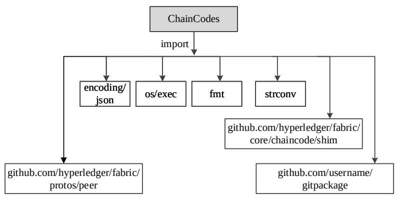

ChainCodes
import
encoding/json
os/exec
fmt
strconv
github.com/hyperledger/fabric/core/chaincode/shim
github.com/hyperledger/fabric/protos/peer
github.com/username/gitpackage

Figure 7 the package dependency relationship of the chaincodes

E. Function Dependency Analysis

Function Dependency Analysis uses inclusion-based pointer analysis to analyze the intermediate representations of the chaincodes, which have the characteristics of the SSA, and constructs Function Call Relationship inside the chaincodes.

Algorithm 2: Function Dependency Analysis

```
begin
    //Select the package with the main function
    mains:=MainPackages(IR)
    //Obtain the original function callgraph
    CallGraph := pointer.Analyze(mains)
    // Traverse all edges of CallGraph
    CallGraph.GraphVisitEdges(){
        // Remove irrelevant callgraph
        CallGraph.RemoveUnwantedEdge()
    }
    //Return the function call relationship needed
    return CallGraph.GetCallGraph()
end
```

The algorithm of the Function Dependency Analysis is implemented as shown in Algorithm 2. The algorithm first analyzes the intermediate representation of the chaincodes to select the packages with the main function. Subsequently, the Analyze function in the pointer library officially provided by the Golang language is used to further analyze the selected package to construct the original function callgraph[17], which contains a lot of call relationships with Edges and Nodes. Then, using the method of depth first search traverses each Edge of the original function callgraph. At the same time, the algorithm removes the irrelevant call Edges, such as shim, peer packages related call edges, the underlying library related call edges, and retains the call relationships useful for subsequent detection. Finally, a clear function call relationship in the chaincodes is constructed.

F. Chaincode Detection Execution

Chaincode Detection Execution undertakes the task of detecting potential risks of the chaincodes. We extract the static structural features of the known risks of the chaincode to form a feature library. Then, the Chaincode Detection Execution matches the static structural features of the chaincodes obtained by the Chaincodes Static Analysis with the feature library to obtain the detection results. It realizes the detection of different risk items through the package detection module, the instruction detection module, and the logic detection module.

The package detection module uses the Depth First Search to search the Package Dependency Relationship of the chaincodes. In the process of searching the Package Dependency Relationship, if a risky package is found, there is a corresponding risk in the chaincodes. Otherwise, if there is a recommended package, there is no risk. The risk items detected by this module, risky packages, and recommended packages are shown in Table I. Since the time packages may introduce the logic of obtaining system time, developers should take care to avoid the non-determinism risk using such packages. In contrast, the crypto/des packages may introduce the function of data encryption, this is a package recommended for developers to increase the protection of private data.

TABLE I
THE DIFFERENT RISKS RELATED TO THE PAKAGES

The risk items	The risky packages
Random Number Generation	crypto/rand, math/rand, ……
System Timestamp	time.Date, time.Now, ……
System Command Execution	os/exec, ……
External Library Calling	not standard libraries , ……
Web Service	net/http, ……
External File Accessing	ioutil, os, ……
The risk items	The recommended packages
Unencrypted Sensitive Data	crypo/md5, crypo/des, ……

The instruction detection module uses the node characteristics of the AST of the chaincodes to determine the potential risks. For example, Global Variables are determined based on the Tok value and ValueSpec value of the ast.Decl node under the root node of the AST. The risk items that this module can detect are shown in Table II.

TABLE II
THE RISK ITEMS DETECTED BY THE INSTRUCTION DETECTION MODULE

The risk items
Global Variable
Field Declarations
Map Structure Iteration
Reified Object Addresses
Concurrency of Program
Cross Channel Chaincode Invocation
Unused Privacy Data Mechanism

The logic detection module detects potential risks according to the path characteristics of the function call relationship inside the chaincodes. The risk items detected by this module include Range Query Risk and Read Your Write. For Range Query Risk, it mainly confirms whether there is a call path from the Invoke function to the data range query function, such as GetPrivateDataQueryResult(), GetQueryResult(). If the path exists, the risk item exists.

IV. EVALUATION

A. Experimental Setup

After the potential risk detection system of the chaincode is designed, we evaluate the efficiency and accuracy of our system. We use a crawler to collect 300 chaincode samples developed by Golang on the Github. These chaincode samples come from different business scenarios, such as quality performance certification, car transactions. Our system runs on a Linux operating system computer, which has an Intel Core i7 CPU and 8.00 GB Ram. The system first detects the collected chaincode samples and counts the number of occurrences of 16 risk items, the number of false positives and the number of false negatives. Then, we calculate the false positive rate (FPR), false negative rate (FNR) and Accuracy of each detection module of the detection system based on the statistical data of the risk items of the chaincode samples. Finally, the false positive rate (FPR), false negative rate (FNR) and Accuracy of the detection system are also calculated.

B. Evaluation Result

Through testing, it was found that 212 chaincode samples in 300 chaincode samples had potential risks, covering all of 16 risk items. This shows that the chaincode developed by the Golang language needs to be tested to discover the potential risks before deployment. At the same time, we calculated the false positive rate (FPR), false negative rate (FNR) and accuracy of each detection module of our system, as shown in Table III. According to the results in Table III, we can see the accuracy of each detection module of our system is higher than 95%, and the false alarm rate and false alarm rate of each detection module are less than 5%, so the detection results have certain reference value. Therefore, our system can assist developers to develop more secure and reliable chaincodes.

TABLE III
THE FPR, FNR, ACCURACY OF EACH DETECTION MODULE

The detection module of our system	FPR	FNR	Accuracy
The package detection module	4.8%	2.5%	96.1%
The instruction detection module	1.8%	4.5%	97.3%
The logic detection module	3.4%	4.4%	95.4%

TABLE IV
RISK ITEMS THAT APPEAR FREQUENTLY

Global Variable	Random Number Generation	Unused Privacy Data Mechanism	Unencrypted Sensitive Data
35.6%	28.2%	40.3%	26.8%

In addition, among the 16 risk items, the following risk items frequently appear in the chaincode samples, the specific results are shown in the in Table IV. These risk items are easy to ignore in the chaincode development process, so developers should pay attention to these risk items that occur frequently, and consider the use of private data mechanism and data encryption more according to specific scenarios.

In the process of detecting chaincode samples, we can analyze the chaincodes and get the analysis result in a few seconds. The user experience is better.

C. Compared with other mainstream tools

The current potential risk detection tools of the chaincode include Chaincode Scanner (CS) and Fujitsu smart contract detection tool. Among them, the CS developed by ChainSecurity is the first smart contract risk detection platform in the world. Our system has been compared with them in terms of covering potential risk items, and the specific comparison results are shown in Table V.

TABLE V
COVERING POTENTIAL RISK ITEMS OF EACH TOOL

The potential risk items	The CS	Fujitsu	Ours
1)Random Number Generation	√	√	√
2)System Timestamp	√	√	√
3)Reified Object Addresses	√	√	√
4)Global Variable	√	√	√
5)Field Declarations	√	√	√
6)Concurrency of Program	√	√	√
7)Map Structure Iteration	√	√	√
8)External File Accessing	×	√	√
9)External Library Calling	×	√	√
10)Web Service	×	√	√
11) System Command Execution	√	√	√
12) Range Query Risk	√	√	√
13) Read Your Write	×	√	√
14) Cross Channel Chaincode Invocation	×	√	√
15) Unused Privacy Data Mechanism	×	×	√
16) Unencrypted Sensitive Data	×	×	√

Through comparison, it is found that our system realizes the detection of common risk items that are mentioned by other researchers, especially, it can detect more potential risk items in terms of the Data Privacy Security Risk, which just makes up for the deficiencies of the other two detection tools, and can guide developers to strengthen the protection of sensitive data.

V. CONCLUSION

In Hyperledger Fabric, this paper focuses on potential risks of the chaincodes developed by the general-purpose programming language, and summarizes 16 risk items, which are divided into three categories: Non-determinism Risk, Logical Security Risk, and Private Data Security Risk. In order to detect different risks of the chaincodes, a new static analysis method is proposed. Compared with the previous static analysis method, this method performs the Package Dependency Analysis and Functional Dependency Analysis on the basis of the abstract syntax tree to obtain static characteristics that can better express different risk items. At the same time, using our static analysis method, we designed a risk detection system of the chaincodes developed by the Golang language, and a lot of experiments have been carried out on the system. The results show that the system can locate the risk location with high accuracy. It also makes up for the shortcomings of the mainstream detection tools of the chaincodes in privacy data risk detection.

With more and more applications of Hyperledger Fabric, potential risky cases of the chaincodes developed by the general-purpose programming language will continue to be discovered, so the feature library of our system needs to be continuously updated. Since our system can only detect the chaincodes developed by Golang language, this is not enough. Our system also needs to update to detect the chaincodes developed by other general-purpose programming languages, such as Nodejs and Java. As the number of risky chaincode samples continues to increase, we can also consider the use of artificial intelligence methods to achieve the detection of potential risks of the chaincode.

REFERENCES

[1] Androulaki E , Manevich Y , Muralidharan S , et al. Hyperledger fabric: a distributed operating system for permissioned blockchains[C]// the Thirteenth EuroSys Conference. 2018.

[2] Foschini L , Gavagna A , Martuscelli G , et al. Hyperledger Fabric Blockchain: Chaincode Performance Analysis[C]// ICC 2020 - 2020 IEEE International Conference on Communications (ICC). IEEE, 2020.

[3] D. Harz and W. J. Knottenbelt. Towards safer smart contracts: A survey of languages and verification methods. CoRR, pages 1–20, 2018.

[4]Gosec.https://github.com/securego/gosec[EB/OL].2019.

[5] Staticcheck.https://staticcheck.io/[EB/OL].2019.

[6] Zhang S, Zhou E, Pi B, et al. A Solution for the Risk of Non-deterministic Transactions in Hyperledger Fabric[C]//2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). IEEE, 2019: 253-261.

[7] Huang Y, Bian Y, Li R, et al. Smart contract security: A software lifecycle perspective[J]. IEEE Access, 2019, 7: 150184-150202.

[8] Yamashita K, Nomura Y, Zhou E, et al. Potential risks of hyperledger fabric smart contracts[C]//2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE). IEEE, 2019: 1-10.

[9] Cachin C. Architecture of the hyperledger blockchain fabric[C]//Workshop on distributed cryptocurrencies and consensus ledgers. 2016, 310(4).

[10]Performance Modeling of PBFT Consensus Process for Permissioned Blockchain Network (Hyperledger Fabric)[C]// 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS). IEEE, 2017.

[11] Ma C, Kong X, Lan Q, et al. The privacy protection mechanism of Hyperledger Fabric and its application in supply chain finance[J]. Cybersecurity, 2019, 2(1): 1-9.

[12] Benhamouda F, Halevi S, Halevi T. Supporting private data on hyperledger fabric with secure multiparty computation[J]. IBM Journal of Research and Development, 2019, 63(2/3): 3: 1-3: 8.

[13] Kitlei R . The reconstruction of a contracted abstract syntax tree[J]. Studia Universitatis Babeș-Bolyai Informatica, 2008, 53(2).

[14] Liu X , Yin W , Yin Q , et al. A SSA-based intermediate representation technique[C]// 2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering. IEEE, 2010.

[15] Varghese S . Using Standard Library Packages[M]. Apress, 2016.

[16] Donovan A , Kernighan B W . Go Programming Language, The[M]. Betascript Publishing, 2015.

[17] Zhhuta V . Go Programing Language (GoLang). 2015.