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

Penghui Lv*†, Yu Wang*,YaZhe 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, zhouqihui}@iie.ac.cn

2021 IEEE Symposium on Computers and Communications (ISCC) | 978-1-6654-2744-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISCC53001.2021.9631249

***Abstract*—As an indispensable part of Hyperledger Fabric application system, smart contracts are mostly developed in general-purpose programming languages such as Golang. However, these smart contracts often have potential risks that cause serious problems such as failed transactions or sensitive information leakage on the ledger. Although there are already some detection tools for potential risks, e.g., Chaincode Scanner and Chaincode Analyzer, the accuracy and coverage of them are limited. In response to the above problems, this paper summarizes 16 potential risks in smart contracts, including three type risks: Non-determinism Risk, Logical Security Risk, and Private Data Security Risk. In order to detect them, we propose a new static analysis method based on Abstract Syntax Tree Analysis, Package Dependency Analysis, and Functional Dependency Analysis. Based on the new method, a detection system is designed that can detect 16 potential risks in smart contracts developed using Golang language more accurately and provide development suggestions to eliminate these risks.**

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

I. Introduction

Hyperledger Fabric is one of the most famous permissioned blockchain [1]. Its smart contract is also called "chaincode", which is a piece of executable code developed by users to define common rules between different organizations. These rules cover common terms, data, and transaction processes. Chaincodes, together with the ledger, form the heart of Hyperledger Fabric application system. Applications invoke chaincodes to generate transactions recorded on the ledger of Hyperledger Fabric.

From an application developer's perspective, 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 security risks to users when these chaincodes are executed, e.g., failed transactions and sensitive information leakage on the ledger of Hyperledger Fabric. 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 and staticCheck [4-5], 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 Hyperledger Fabric. Zhang et al. [6] summarizes the non-deterministic risks in the chaincodes of Hyperledger Fabric. Huang Y et al. [7-8] introduced Chaincode scanner, which can only detect 9 risks related to the non-determinism risks and logical security risks in the chaincodes. Yamashita K et al. [9] designed a Chaincode Analyzer based on the Abstract Syntax Tree, this tool can detect 14 risks mainly related to the non-determinism risk and logical security risk in the chaincodes.

Although the above research has summarized some risks related to chaincode, the introduction of these risks is not comprehensive enough. In addition, current tools detect potential risks in the chaincodes only based on the Abstract Syntax Tree, the accuracy and coverage of these tools are limited. In particular, these tools have shortcomings in detecting potential risks related to private data security. These risks are also often ignored by developers, but most users are concerned.

In response to the above problems, this paper makes the following contributions.

- This paper summarizes 16 potential risks in the chaincodes, and groups them into three types: Non-determinism Risk, Logical Security Risk, and Private Data Security Risk.
- For detecting different risks of Hyperledger Fabric smart contract, a new static analysis method is proposed, which is to perform Abstract Syntax Tree Analysis, Package Dependency Analysis and Function Dependency Analysis on the chaincodes to obtain the detailed static characteristics of different potential risks.
- A potential risks detection system of the chaincodes is designed based on the new static analysis method. It can detect 16 potential risks in smart contracts developed using Golang language more accurately, of which the Data Privacy Security Risks are detected, and makes up for the deficiencies of other tools.

The rest of our paper is organized as follows. 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 experiments conducted on our system, and then compared our system with other tools. 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 [10], as shown in Fig. 1.

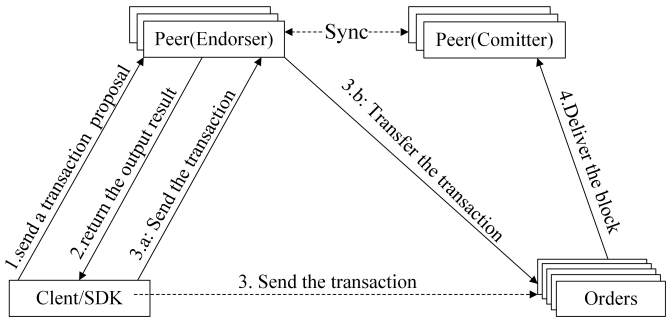

Fig. 1. The transaction flow of Hyperledger Fabric.

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 paper 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

In Hyperledger Fabric, transaction failure caused by Non-deterministic Risk is reflected in the transaction endorsement

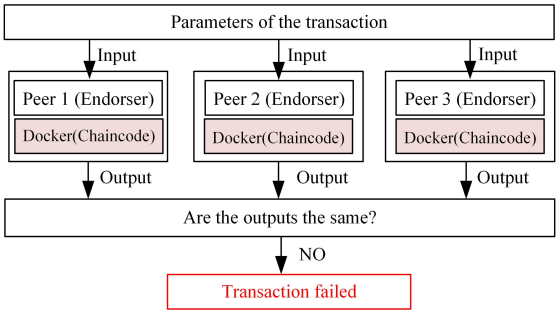

Fig. 2. The process of failed transaction caused by non-deterministic risk.

step, as shown in Fig. 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 [11]. 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.

Listing 1: Example of Non-deterministic Data Sources

```
1  begin
2    // User set a value
3    setValue := arg[0]
4    rand.Seed(seed) // random function
5    sel := rand.Intn(10)
6    if setValue == sel{a → b}
7    else {a ← b}
8  end
```

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 Fig. 3.

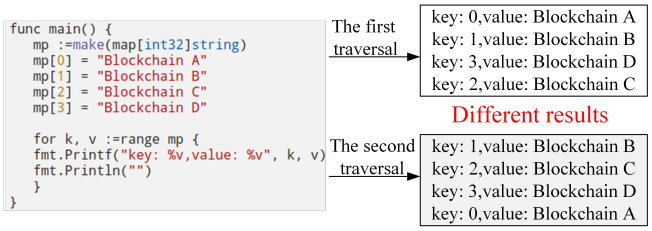

Fig. 3. Non-deterministic results coming from Map Structure Iteration.

Definition 3: Non-deterministic External Calls refer to accessing outside of the blockchain which can 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 of the chaincodes with Non-deterministic External Calls running in different peers are not guaranteed to be consistent. For example, there is a risk of Web Service in the chaincodes. When different peers execute these chaincodes to obtain data from the website, this website may return different data to each peer. Therefore, when encountering Non-deterministic External Calls, developers need to be clear about what results from the outside of the blockchain 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 a potential risk about Unencrypted Sensitive Data in the chaincodes, the plaintext of sensitive transaction data is stored on 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 [12-13]. 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() [3]. These methods are executed during the transaction endorsement phase, but are not re-executed during the transaction validation 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.

Listing 2: Example of Read Your Write Risk

```
1  begin
2    // At the initial point: key: "key", value: 0
3    value := 1
4    // Update the value from 0 to 1
5    err := stub.PutState("key", value)
6    // Return 0, not 1
7    ret := stub.GetState("key")
8  end
```

Listing 2 shows an example of Read Your Write Risk. The initial value of the key is "0" in the ledger. After the called PutState function will update the key value, it is predicted that the key value is "1" at this time, but in fact the key value read by the GetState function is still "0". Hence, Read Your Write is not supported in Hyperledger Fabric.

III. Design and Implementation of Our System

A. Architecture of Our System

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

1) Chaincode Static Analysis. In this module, the chaincodes of Hyperledger Fabric are analyzed to obtain static

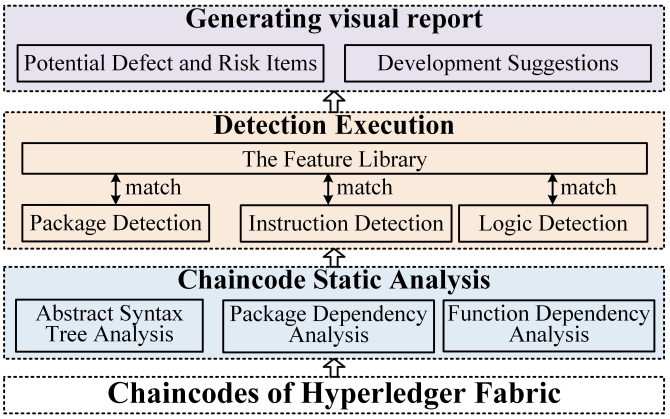

Fig. 4. Architecture of our system.

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 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 risks in the chaincodes, and development suggestions to eliminate these risks.

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 Fig. 5.

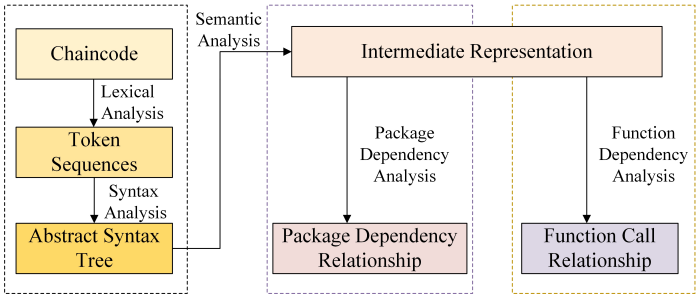

Fig. 5. Simplified diagram of Chaincode Static Analysis.

1) 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 [14].

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. The main part of the abstract syntax tree of chaincodes is shown in Fig. 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.

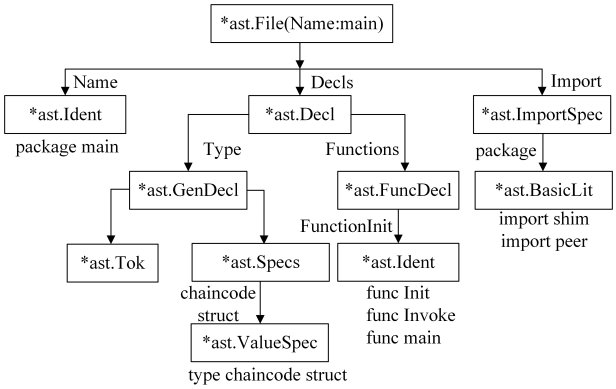

Fig. 6. Abstract Syntax Tree of the chaincodes.

2) 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)
6     processPkg(root, pkgName, level, imported ) {
7       // Level is not greater than maxLevel
8       if level++; level > ∗maxLevel { return nil; }
9       // Get the dependent package list of this layer
10      pkg := buildContext.Import(pkgName, root)
11      //Get the path of the dependent package
12      importPath := normalizeVendor(pkgName)
13      pkgs[importPath] = pkg
14      // Recursive
15      for _, imp := range getImports(pkg) {
16          if _ , ok := pkgs[imp]; !ok {
17          processPkg(pkg.Dir, imp, level, pkgName) }
18        }
19      }
20      return pkgs;
21  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) [15]. 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 [16-17]. Finally, the Package Dependency Relationship of the chaincodes is obtained, as shown in Fig. 7.

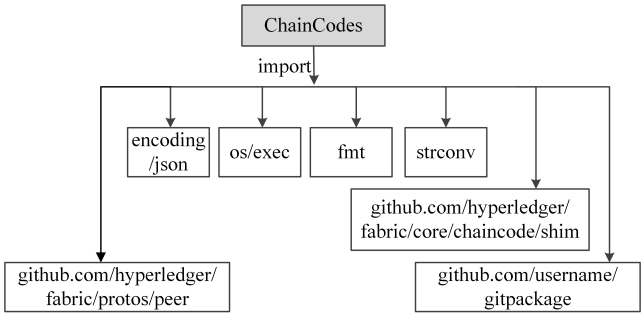

Fig. 7. Package Dependency Relationship of the chaincodes.

3) 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

```
1   begin
2     // Select the package with the main function
3     mains:=MainPackages(IR)
4     // Obtain the original function callgraph
5     CallGraph := pointer.Analyze(mains)
6     // Traverse all edges of CallGraph
7     CallGraph.GraphVisitEdges(){
8       //Remove irrelevant callgraph
9       CallGraph.RemoveUnwantedEdge()
10    }
11    //Return the Function Call Relationship needed
12    return CallGraph.GetCallGraph()
13  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 [18], 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, Function Call Relationship in the chaincodes is constructed.

C. 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 risks through the package detection module, the instruction detection module, and the logic detection module.

TABLE I
THE POTENTIAL RISKS RELATED TO THE RISKY OR RECOMMENDED PACKAGES

The Potential risks	**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 potential risk	**The recommended packages**
Unencrypted Sensitive Data	crypo/md5, crypo/des, ...

1) Package 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 corresponding risk.

The potential risks detected by this module and the corresponding risk or recommended packages are shown in Table I. Since the time packages may introduce the logic of obtaining system time of different peers, there is the risk of System Timestamp using the time packages. developers should try not to use this package to avoid the risk from System Timestamp. When a business logic in the chaincodes obtains website data outside the blockchain using the net/http package, this website may return different data to each peer, so the net/http package can cause the risk of Web Service in Non-deterministic External Calls. Therefore, it is also necessary to remind the developer. However, The crypto/des packages may introduce the function of data encryption to increase the protection of

private data. If there is a crypto/des package, there is no risk of Unencrypted Sensitive Data.

2) Instruction detection module: The instruction detection module uses the node characteristics of the AST of the chaincodes to determine the potential risks. The potential risks detected by this module include Global Variable, Field Declarations, Map Structure Iteration, Reified Object Addresses, Concurrency of Program, Cross Channel Chaincode Invocation, and Unused Privacy Data Mechanism. Among them, Global Variables are determined based on the Tok value and ValueSpec value of the ast.Decl node under the root node of the AST.

3) Logic detection module : The logic detection module detects potential risks according to the path characteristics of the function call relationship inside the chaincodes. The potential risks 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 potential risk exists.

D. Generating Visual Report

The Generating Visual Report module is designed to show the detection results from the Chaincode Detection Execution to the users in the form of a visual report. The report shows each potential risk and its location in the chaincodes, and provides suggestions to eliminate these potential risks.

IV. Evaluation

After our system based on our proposed new static analysis method was designed, we conducted some experiments to evaluate the usefulness, accuracy and coverage of our system. In terms of accuracy and coverage, we compared our system with the existing tools such as Chaincode Scanner (CS) [8] and Chaincode Analyzer(CA) [9]. The detailed experiment and its results are as follows.

A. Experimental Setup

Our system runs on a Linux operating system computer, which has an Intel Core i7 CPU and 8.00 GB RAM. Since there were no ready-made or standard chaincode samples developed by Golang, we have used a crawler to collect 300 chaincode samples on the Github. These chaincode samples come from different business scenarios, e.g., quality performance certification and car transactions. Then, we used our system to detect the collected samples one by one.

B. Evaluation Result

1) Usefulness of our system: In the process of detecting chaincode samples, we counted the number of potential risks in these samples, the number of false positives, and the number of false negatives. Since these potential risks are detected by different detection modules of our system, we also calculated the false positive rate (FPR), false negative rate (FNR) and accuracy of each detection module of our system based on the above experimental data.

Through the experiment, it is found that 212 out of 300 chaincode samples have potential risks, covering all of 16 potential risks summarized in Section II. The number of each potential risk in 300 chaincode samples is shown in Fig. 8. According to the counted data in Fig. 8, some potential risks that occur frequently are risks that are easily ignored by developers, e.g., Unused Privacy Data Mechanism, Unencrypted Sensitive Data, and Global Variable. It is necessary for chaincode detection tools to detect them to remind developers.

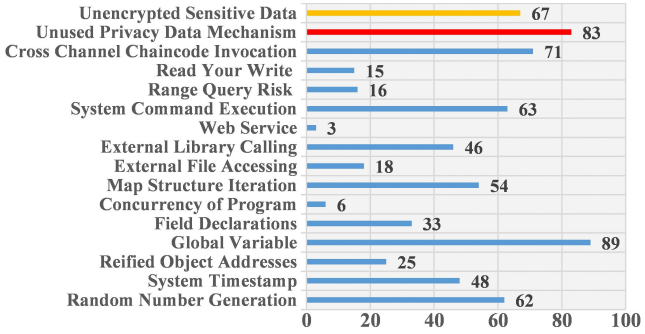

Fig. 8. Potential risks detected by our system and their number.

To further prove the usefulness of our system, we also calculated the false positive rate (FPR), false negative rate (FNR) and accuracy of each detection module of our system, as shown in Fig. 9. From Fig. 9, we can see the accuracy of each detection module of our system is higher than 95%, and the false positive rate and false negative rate of each detection module are less than 5%, the detection results have certain reference value. Therefore, our system can detect 16 potential risks to help developers improve the quality of chaincodes.

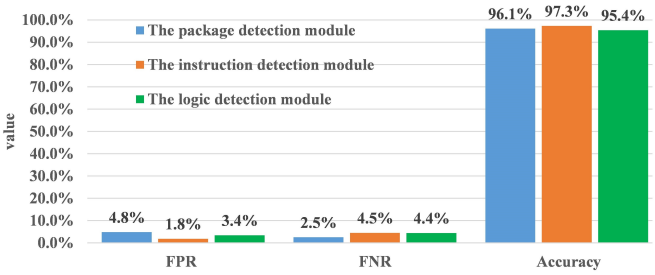

Fig. 9. The FPR, FNR, Accuracy of each detection module of our system.

In the process of detecting each chaincode sample, our system can generate a visual report in a few seconds. The user experience is also better.

2) Accuracy of different tools: By detecting 300 chaincode samples of this experiment with Chaincode Scanner (CS) [8], Chaincode Analyzer(CA) [9] and our system, we obtained the accuracy of these tools, as shown in Fig. 10. By comparison, the accuracy of our system is the highest, which is 8.8% higher than that of Chaincode Analyzer. So our system can assist developers to develop more secure and reliable chaincodes. It also proves that our proposed method has certain advantages.

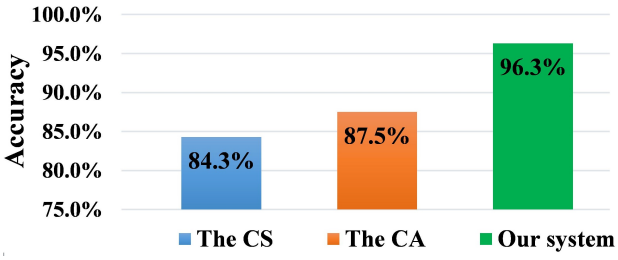

Fig. 10. The accuracy of each tool.

3) Coverage of different tools: The potential risks covered by Chaincode Scanner(CS) [8], Chaincode Analyzer(CA) [9] and our system were counted, as shown in Table II. It is found that our system, which can detect 16 potential risks, has a larger coverage than the Chaincode Scanner and Chaincode Analyzer. It can especially detect the risks of Unused Privacy Data Mechanism and Unencrypted Sensitive Data and guide developers to strengthen the protection of sensitive data.

TABLE II
COVERING POTENTIAL RISKS OF EACH TOOL

The potential risks	The CS	The CA	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	×	×	✓

‘ ✓’means that this potential risk is covered.
‘ × ’ means that this potential risk is not covered.

V. CONCLUSION

In Hyperledger Fabric, this paper focuses on potential risks of the chaincodes developed by the general-purpose programming language, and summarizes 16 potential risks, which are divided into three categories: Non-determinism Risk, Logical Security Risk, and Private Data Security Risk. In order to detect this risks, 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 based on the abstract syntax tree to obtain static characteristics that can better express different risks. 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 other 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.

VI. ACKNOWLEDGMENTS

This work is supported by the National Key R&D Program of China (No. 2019YFB1706000). This work was completed under the guidance of my Ph.D. supervisor, Zhen Xu. Master Han Wang and Ya Chen also participated in this work.

REFERENCES

[1] Androulaki E, Barger A, Bortnikov V, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains[C]. Proceedings of the thirteenth EuroSys conference. 2018: 1-15.
[2] Foschini L, Gavagna A, et al. Hyperledger Fabric Blockchain: Chaincode Performance Analysis[C]. ICC 2020-2020 IEEE International Conference on Communications (ICC). IEEE, 2020: 1-6.
[3] Harz D, Knottenbelt W. Towards safer smart contracts: A survey of languages and verification methods (2018)[J]. 1809.
[4] Gosec.https://github.com/securego/gosec[EB/OL]. 2019.
[5] Staticcheck.https://staticcheck.io/[EB/OL]. 2019.
[6] Zhang S, Zhou E, 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] T. Kaiser. Chaincode Scanner: Automated Security Analysis of Chaincode. ChainSecurity. Accessed: Sep. 6, 2019. [Online]. Available: https://chainsecurity.com/audits.
[9] Yamashita K, Nomura Y, et al. Potential risks of hyperledger fabric smart contracts[C]. 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE). IEEE, 2019: 1-10.
[10] Cachin C. Architecture of the hyperledger blockchain fabric[C]. Workshop on distributed cryptocurrencies and consensus ledgers. 2016, 310.
[11] Sukhwani H, Martínez J M, et al. Performance modeling of PBFT consensus process for permissioned blockchain network [C]. 2017 IEEE 36th Symposium on Reliable Distributed Systems. IEEE, 2017: 253-255.
[12] 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.
[13] 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.
[14] Kitlei R. The reconstruction of a contracted abstract syntax tree[J]. Studia Universitatis Babeș-Bolyai Informatica, 2008, 53(2).
[15] 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, 6: 98-101.
[16] Varghese S. Using Standard Library Packages[M]. Go Recipes. Apress, Berkeley, CA, 2016: 103-127.
[17] Donovan A, Kernighan B W. The Go programming language[M]. Addison-Wesley Professional, 2015.
[18] Zhhuta V. Go Programing Language (GoLang). 2015.