Attack Graph Auto-Generation for Blockchains based on Bigraphical Reaction Systems

Lian Yu
School of Software and Microelectronics
Peking University
Beijing, China
lianyu@ss.pku.edu.cn

Lijun Liu
Institute of Research and Development
China Mobile
Beijing, China
ljliu@mobile.cn

Yanbing Jiang
School of Software and Microelectronics
Peking University
Beijing, China
ybjiang@pku.edu.cn

Qi Jing
School of Software and Microelectronics
Peking University
Beijing, China
qijing@pku.edu.cn

Bei Zhao
Institute of Research and Development
China Mobile
Beijing, China
bzhao@mobile.cn

Chen Zhang
Institute of Research and Development
China Mobile
Beijing, China
czhang@mobile.cn

***Abstract*—Blockchains (BCs) are claimed to have immutability, distributed consensus, established trust, distributed identity and eternal verifiable, and sound like the ultimate security unimpeachable technology. At the time, however, new age security attacks on the key components of BCs are emerging, which are very sophisticated and can cause huge irreparable damages, including network-based attacks, consensus & ledger-based attacks, smart contract-based attacks, and wallet-based attacks. This paper proposes to use bigraph theory to model BC attack meta-model, and automatically generate attack graphs for BC security evaluation. Bigraphical sorting mechanism is used to depict configuration of BC systems, and bigraphical reaction rules are designed to characterize attack templates and attacker behaviours. Adaptive exploit flow approach is proposed to reduce the complexity of matching algorithm guided by interested attack exploits, and probability is introduced into bigraphs to measure the capability of attackers. Preliminary experiments have shown the validity of the proposed approach.**

***Index Terms*—attack graph generation, bigraphical reaction rules, blockchain network.**

I. Introduction

Bitcoin [23] is a rare case where practice seems to be ahead of theory. There are tremendous opportunities for the research community to tackle the many open questions about Bitcoin, especially its pillar technology, blockchain.

Blockchains (BCs) are considered to have immutability, distributed consensus, established trust, distributed identity and eternal verifiable claims, and the technology has been expected to be applied to many fields. However, a series of cyber-attacks against digital currencies has left the important services industry, such as finance, insurance, charity, and government sectors, wondering whether new blockchain technology can be made secure enough from criminals. It is indispensable to carry out the security risk analysis on BC systems to evaluate how much the systems can achieve its promises and intentions.

The research was partially supported by Ministry of Education-China Mobile (MCM20170406), the National Natural Science Foundation of China (No. 61872011), and the Ministry of Science and Technology of China (No. 2018YFB1402700).

A blockchain is composed of four key modules in terms of its hierarchy from the system's point of view:

- Peer-to-peer network and high-speed network;
- Consensus engine and distributed ledger;
- Scripting and virtual execution environment for smart contracts;
- Wallet applications

When the target of BC attacks is Peer-to-peer network or high-speed network, the attackers may launch Eclipse attack [49] and Sybil attack [21]. In a P2P Network-based blockchains, a node will depend on ”*x*” number of nodes selected using a Peer selection strategy to have its view of the distributed ledger. But if an attacker can manage to make the node to choose all the ”*x*” number of nodes from the attacker's malicious nodes alone, then the attacker can eclipse the original ledger's view and present his own manipulated ledger to the node.

Sybil attack targets the whole network, instead of eclipsing a user's view of the true ledger as Eclipse attack does. In a Sybil attack, an attacker will flood the network with a large number of nodes with pseudonymous identities and try to influence the network. Both Eclipse attack and Sybil attack aim at gradually getting control over the blockchain system, eventually achieving double spending or obtaining illegal assets.

When the target of BC attacks is consensus engine and distributed ledger, there are more instances in category of consensus mechanism and mining-based attacks, including selfish mining attack [41], mining malware [42], 51% attack [43], time-jack attack [44], Finney attack [45], and race attack [46]. For example, many blockchains consider the longest chain to be the true latest version of the ledger, and a selfish miner can try to keep building blocks in stealth mode on top of the existing chain, and when he can build a lead of greater than two or more blocks than the current chain in the network, he can publish his private fork, which will be accepted as a new truth as it is the longest chain. 51% attack is possible when a miner or a group of miners controls 51% or more of the mining power of the blockchain network. Though it is very difficult to happen for large networks, the possibility of a 51% attack is higher in small networks. Once a group has majority control over transactions on a blockchain network, it can prevent specific transaction or even reverse older transactions. These types of attacks exploit the consensus mechanisms, and eventually conquer blockchain systems.

Smart contracts are completely automated contracts, which execute transactions in an agreed upon way between participants, with inputs from the real world and without intervention from any middlemen. Once started, a smart contract cannot be stopped. The transaction once completed and written into blockchain becomes immutable. This gives a guarantee to participants of returns based on their performance, as agreed upon while entering the contract. But what would happen if the smart contract has bugs: Millions of dollars are in stake and no one can change it. The DAO attack [47] is such an attack or attack vector relating to smart contracts. The program vulnerabilities of smart contracts are exploited, thus leading to huge irreparable damages.

Wallets act as clients in blockchain systems. Parity multisig wallet attack [48] was the case of a vulnerability with the parity client wallet hacked by an attacker resulting in holding up of 500,000 Ether. Multisig wallet functionality (Multisig wallet is like a joint account in bank with multiple owners) used a centralized library contract. However, they left some critical functions open, resulting in a vulnerability, which was exploited by the attacker. Wallet attacks lead to huge losses of user assets.

Blockchain systems have more complex decentralized network structures, consensus hosts don′t trust with each other, and independently run agreed up smart contracts. Analysing and defending such large and complex blockchain networks both from insides and outsides attacks is an uphill task. This paper would develop an approach to performing security assessment and evaluation to protect blockchains from those attacks.

Attack graph [34] is a modelling tool, representing all attack scenarios in a visual data structure, and serve as a basis for risk analysis, defence, detection, and forensics. An exhaustive attack graph of a network provides omni-bearing view of its security posture, enabling the quantitative assessment. Such assessments, when performed on-line, help a blockchain network system to thwart the attacks by removing off the preconditions of the attacks. This will prevent blockchain attacks, such as 51% attacks.

Major approaches to attack graph generation include state enumeration based approaches, topological vulnerability analysis (TVA) [16], logic programming based approach and network security planning architecture approach [4]. Among the rest, state enumeration based approaches are based on model checking. TVA requires an extensive knowledge base of known vulnerabilities and attack techniques. One requirement of logical attack graph is that the cause of an attacker's potential privilege should be expressible as a propositional formula in terms of network configuration information.

The issues with current research approaches when applied to generate attack graphs for blockchain systems are summarized as follows:

- Lacking a meta-model: Different approaches have different advantages, but don′t provide a meta-model to depict the common or shared characteristics of network domains, such as blockchain network;
- A variety of types of attack graphs: Different approaches define and develop different types of attack graphs, including scenario graphs, exploit dependency attack graph, logical attack graph, and multiple-prerequisite attack graph. This paper only uses bigraphs to build a meta-model, create blockchain attack models, and auto-generate attack graphs;
- Strong assumptions: To improve the performances of attack graph generation, different approaches present different solutions with different assumptions. Those assumptions may be no longer valid in blockchain network, such as the monotonicity assumption [1], and need to develop an adaptive solution to deal with.

This paper proposes to apply bigraph theory to model and automatically generate attack graphs. The contributions of this paper regarding attack graph generation are summarized as follows:

- Build a meta-model of blockchain attacks: The physical world of blockchain network and its vulnerabilities are abstracted and mapped to bigraphical signature and sorting mechanism as shown in Figure 1;
- Create reaction rules and bigraphs: A set of bigraphical reaction rules are constructed to depict the attack templates of attackers, and a set of place graphs and link graphs to describe attacker's profiles and configuration;
- Model uncertainty of attacker's behaviours: The paper extends the expression capability ofbigraphical reaction rules by accommodating the probabilities of blockchain attacks into the rules.
- Optimize performance with adaptive exploit flow: In BC network, the monotonicity assumption is no longer valid. To alleviate the burden of computation, this paper defines attack exploit flow to generate the attack graph efficiently, instead of taking the advantages of monotonicity assumption.

The rest of the paper is organized as follows. Section II gives a brief introduction to bigraphs and bigraphical reactive systems (BRSs). Section III describes modelling of BC network vulnerabilities using bigraph theory. Section IV illustrates attack graph generation with execution of BRSs, and compares with previous work. Section V describes the optimization of the proposed approach. Section VI presents the experimental results with the developed tools by the authors'

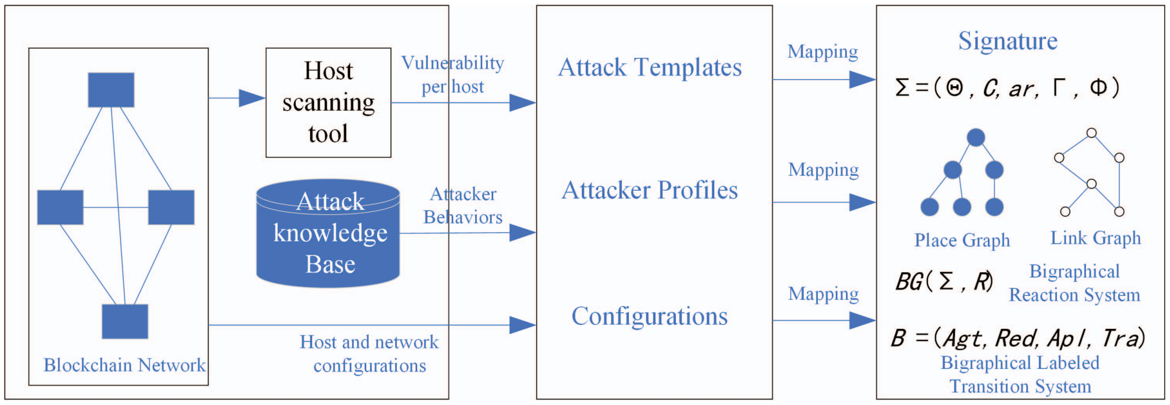


Fig. 1. From a Physical World to Bigraphs

team and Section VII discusses related work. Section VIII gives the conclusions of the paper.

II. Bigraphs and Bigraphical Reaction Systems

The section introduces the static structure of bigraphs, dynamics of BRS, and bigraphical operations, which will be used to model the conference of blockchain network, attacker's behaviours and constructions of blockchain network for simulation and analysis.

A. Modelling Static Structures with Bigraphs

The two typical traits of structures in a blockchain network are containment and connections. A blockchain network consists of a set of validation nodes, each of which contains a set of hosts, and in turn, each of which are installed a variety of software. This forms a hierarchical structure. On the other hand, the elements in a blockchain network are connected with each other, e.g., programs are communicating by sending and receiving messages, and hosts are linked together by cables. An attacker uses a terminal to launch an intrusion attack. These are the examples of connections. A bigraph can be used to describe such structures.

A bigraph represents orthogonal notions of locality and connectivity through two separate graph structures: a place graph and a link hyper-graph. It is formally defined as follows.

Definition 1 (Bigraph). A bigraph is

$$\mathcal{B} = (V_{\mathcal{B}}, E_{\mathcal{B}}, ctrl_{\mathcal{B}}, prnt_{\mathcal{B}}, link_{\mathcal{B}}) : \\ < m, X > \rightarrow < n, Y > \quad (1)$$

where a pair $\langle j, X \rangle$ represents an interface where $j \geq 0$, indicating the number of sites or roots where a site is a placeholder that can be replaced with a root, and X is a set of (inner or outer) names; $V_{\mathcal{B}}$ is a set of nodes, $E_{\mathcal{B}}$ is a set of hyper-edges, $ctrl_{\mathcal{B}}$ is a control map that assigns controls to nodes, $prnt_{\mathcal{B}}$ is a parent map that defines the tree structure in the place graph, and $link_{\mathcal{B}}$ is a link map that defines the link structure. The inner interface $\langle m, X \rangle$ indicates that the bigraph has $(m + 1)$ sites, and a set of inner names X. The outer interface $\langle n, Y \rangle$ indicates that the bigraph has $(n + 1)$ roots and a set of outer names Y.

Bigrah $\mathcal{B}$ consists of a place graph $\mathcal{B}^P = (V_{\mathcal{B}}, ctrl_{\mathcal{B}}, prnt_{\mathcal{B}}) : m \rightarrow n$, and a link graph $\mathcal{B}^L = (V_{\mathcal{B}}, E_{\mathcal{B}}, ctrl_{\mathcal{B}}, link_{\mathcal{B}}) : X \rightarrow Y$, and written as $\mathcal{B} = \langle \mathcal{B}^P, \mathcal{B}^L \rangle$. The bigraphs will be used to build the static structures for generating attack graphs as described in Section III.A.

Definition 2 (Signature Σ): A signature Σ for a bigraph is represented as a quintuple:

$$\Sigma = (\Theta, C, ar, \Gamma, \Phi) \quad (2)$$

where $\Theta = \{\Theta^P, \Theta^L\}$ is a non-empty set of sorts, Θ^P is the place sorts, and Θ^L is the link sorts; $C = \{c_1 : \theta_1, \ldots, c_k : \theta_k, \ldots, c_n : \theta_n\}$ is a set of sorted controls, $\theta_k \in \Theta^P$ indicating the place sort of control c_k ; $ar : C \rightarrow \omega$ is a function assigning a finite ordinal (the arity) to each control, and $ar(c_k) = ar_k; \Gamma : \omega \rightarrow \Theta^L$ is a function assigning one sort to each arity, that is, for control c_k , the ith port is assigned sort $\theta_i \in \Theta^L$; Φ is the formation rules, and represented as sorting logic to provide some constraints on bigraphs.

The bigraphical signature as to bigraphs is similar to the Object Constraint Language (OCL) as to UML, which appeared as an effort to overcome the limitations of UML when it comes to precisely specifying detailed aspects of a system design [8]. OCL has now become a key component of model-driven engineering technique for expressing model transformations, well-formed rules, or code-generation templates. The bigraphical signature is the built-in mechanism in bigraphs to express the constraints and rules on containments, connections, the number of ports, port types, logic relations among the elements. It supports to define and create new domain-specific meta-model, and facilitates to build domain models, and performs simulations and analysis. This paper abstracts the common features of blockchain attacks and uses the bigraphical signature to build a meta-model for auto-generation of attack graphs, as described in Section III.A.

In his book [20], Milner defines operations of composition, juxtaposition, parallel and merge on bigraphs, and defines signatures and sorting logic to represent constraints on bigraphs for specific domains. These operations can be used to build up a blockchain network from scratch in the simulation, and add and remove nodes during the processing.

B. Modelling Dynamic Behaviours with BRSs

Taken together, a bigraph signature and a set of reaction rules is called a bigraphical reactive system that should be thought of as a kind of language definition, with the signature describing the syntax, and the reactions describing the semantics.

Definition 3 (Bigraphical Reactive System, BRS). The notation BG(Σ, R) is used to denote a bigraphical reactive system with a signature Σ (the set of constituent controls), and a set of reaction rules R.

Each reaction rule has two portions at two side of $\rightarrow$, the left of the arrow is called redex and the right reactum. One can understand that the redex is the state before the reaction rule firing, and the reactum is the state after the reaction rule firing. This paper uses bigraphcal reaction rules to model the attack templates, and attacker's behaviours as described in Section III.B.

Definition 4 (Bigraphical Labelled Transition System, B-LTS): A bigraphical labelled transition system is a quadruple:

$$B - LTS = (Agt, Red, Apl, Tra)$$

where Agt is a set of agents, Red is a set of redexes, $Apl \subseteq Agt \times Red$ is the applicability relation, $Tra \subseteq Apl \times Agt$ is the transition relation. One can understand that if an agent and a redex meet the composition operation condition, the reaction rule can be fired, then the agent and the redex have the applicability relation, apl, $ap \in Agt \times Red$; after the firing or transition, tra, a new agent is obtained, $tra \in Apl \times Agt$.

Definition 5 (Attack Graph): A attack graph or *AG* is a tuple $G = (Agt, Red, Apl, Tra, Agt_S)$, where $Agt_S \in Agt$ is a set of attack success agents.

Bigraph-based attack graph allows to analyse risks to blockchain network in a way with more refined granularities by taking advantages of bigraph modelling capability. The following section will present modelling attack graphs with bigraphs.

III. Modelling Attack Graphs with Bigraphs

The attack graph can be automatically generated given three types of inputs: attack templates, a configuration file, and an attacker profile [1]. This section describes the approaches to modelling configurations with bigraph agents and attack templates with bigraphical reaction rules.

A. Building a static structure for generating attack graphs in BC

The general signature, $\Sigma = (\Theta, C, ar, \Gamma, \Phi)$ defined in Section II is instantiated to adapt modelling BC attacks to generate the attack graphs. Let $\Theta^P = \{validationNode, host, software, vulnerability, attacker, priviledge\}$ be the place sort, to specify the validation nodes performing consensus protocols, hosts carrying out tasks of validation nodes, software installed, vulnerability of BC networks, attackers and privileges in BC.

Let $\Theta^L = \{connect, install, hasVul, isVul, exploit, hasPri, hasValue\}$ be the link sort, presenting binary or unary relations, including connections, software installation, having vulnerability, is-a vulnerability, attacker's exploits on BC, someone having a privilege, and having some value; and $C = \{VNode : validationNode, Host : host, SW : software, Vul : vulnerability, AT : exploit, Pri : priviledge\}$ be the set of sorted controls to indicate different types of controls defined by the place sorts.

$ar : C \rightarrow \omega$ is a function assigning a finite ordinal (the arity) to each control, and the $\Gamma : \omega \rightarrow \theta^L(\theta^L \in \Theta^L)$ is a function assigning a link sort to each port of a control, i.e., $\Gamma \circ ar = \{VNnode* : connect; install, SW* : install; hasVul; exploit, Vul* : IsVul, AT* : hasPri, Pri* : hasValue\}$, indicating that the controls $VNode, SW, Vul$ and AT have $0 \sim n$ number of ports, and each port can be assigned the corresponding sorts after the colon, separated by a semicolon. For example, the control SW has $0 \sim n$ number of ports with either $install$, or $hasVul$, or $exploit$ sorts.

There are a set constraints in Φ, among them, some are constraints on Θ^P , and the rest on Θ^L, represented in sort logic [28]. The following presents four examples:

θ_1^P**:** The children of *VNode* nodes must be *Host* nodes:

$$VNode(u) \wedge u.v \Rightarrow Host(v)$$

θ_2^P**:** The children of *Host* nodes must be *SW* nodes:

$$Host(u) \wedge u.v \Rightarrow SW(v)$$

θ_1^L**:** A *connect* port i of a permissioned node u can be only linked to *connect* port j of another permissioned node v:

$$VNode(u) \wedge u@i : connect = VNode(v) \wedge v@j : connect$$

θ_2^L**:** The ***hasPri*** port i of an attacker node can be only linked to ***hasValue*** port j of a privileged node:

$$AT(u) \wedge u@i : hasPri = Pri(v) \wedge v@j : hasValue$$

B. Modelling configuration with agents

The configuration gives detailed information about BC system to be analysed including the topology of the network connecting among participates and configuration of network elements [1], such as validation nodes in the consortium blockchain system, consensus algorithm modules installed as shown on the right in Figure 2, where there are four validation nodes (b:Node through e:Node) that are connected through specified ports, and permitted to participate consensus voting in a consortium blockchain system.

The attacker profile contains information about the assumed attacker's capabilities, with a port on the attacker control, to indicate its possession as well as skill level as shown on the

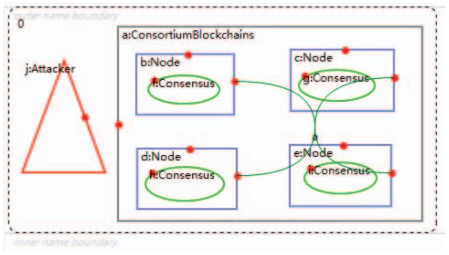


Fig. 2. Modelling configuration of BC system

left in Figure 2, where the attacker has one port to connect to a privilege level.

In addition to the formalism and visual structures, bigraphs can be represented in textual term language [3], and a $B{-}LTS$ can be generated by a term rewriting approach that applies rewriting on reaction rules to enable a system with dynamic behaviour. The following is the corresponding formula in Term language, an internal expression of bigraph processed by a matching algorithm.

```
# Controls
%active Greater:2; Less:2; GreaterOrEqual:2;
%active LessOrEqual:2; Equal:2;NotEqual:2;
%active Exist:1; InstanceOf:2;
%active Attacker:1; ConsortiumBlockchain:1;
%active Node:2;Consensus:1;

# Model
%agent a:Attacker[idle]
b:ConsortiumBlockchain[idle].
(d:Node[idle,a:edge].g:Consensus[idle]|
c:Node[idle,a:edge].i:Consensus[idle] |
e:Node[idle,a:edge].h:Consensus[idle] |
f:Node[idle,a:edge].j:Consensus[idle]);
```

It is noted that the shapes and colors of controls, nodes and edges have no semantics regarding modelling BC systems and attacks, but just for visual purposes.

C. Modelling attack templates and attacker behaviours with rules

Attack templates represent generic or hypothesized attacks including conditions, such as operating system version, which must hold for the attack to be possible. Bigraphical reaction rules represent the dynamics of the system analysed, reflecting the configuration changes and attacker behaviours. Figure 3 shows a reaction rule where the left of the arrow is the *redex*, the same as Figure 2, and the right is the *reactum* indicating that an attacker takes an action, pretends a legal IP and smoothly gets through firewall of a consortium blockchain system, and obtains higher probability to further attack the system.

Upon the rules, one can also specify the condition of reaction with Boolean expression, the input/output batch data, the assignment of variables, probabilities of reaction rules, and timing mechanism as shown at the bottom in Figure 3 .

After the attacker goes through the firewall, the attacker launches attacks to all validation nodes, as shown in Figure 4, where red edges in the reactum indicate sniffing ports on these nodes. As a result, the attacker successfully attacks a node, e.g., *b*, while the rest of other nodes survive. Furthermore, the attacker hacks the consensus module within the node. Eventually, the attacker makes the node drop out of voting. The attacker may choose DOS attack, and this causes the time-out for the node to send messages to other nodes, and achieve the same purpose. Each possible step of attacks can be represent as a reaction rule.

If the consortium applies Byzantine fault tolerant (BFT) algorithm [9], BC system still works normally even there is one node attacked out of four nodes, i.e., BC system still creates blocks and stores in distributed ledgers (DL) thanks to the fault-tolerance of BFT. However, when there are two consensus modules on two *v*-nodes (i.e., validation nodes) attacked and curbed, the BC system could not produce blocks any more, i.e., in a paralysed situation. Unfortunately, when three consensus modules on three *v*-nodes are overcame, the BC system is totally under the control of the attacker, as the assumption of BFT is violated and the guarantee no longer keeps true.

IV. Generating Attack Graphs with BRS

This section presents a matching algorithm to generate bigraphical labelled transition system (*B-LTS*, see Definition 4) that can characterize attack graphs, and describes the architecture of the proposed approach to fulfil the tasks.

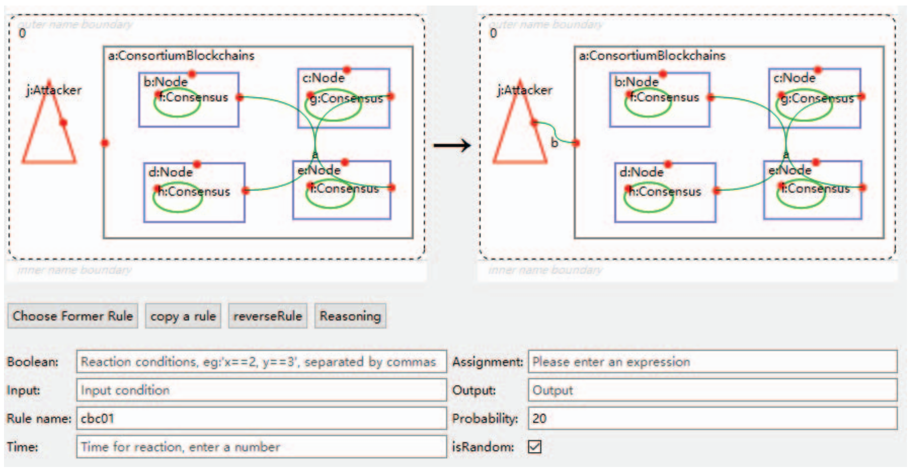
A. Bigraphical Matching Algorithm

This paper uses the term language in the matching algorithm to perform bigraphical pattern matching and carry out the bigraphical simulation. The matching process determines the presence of a redex of reaction rule r inside a particular bigraph A, and replaces the redex with the reactum, obtaining a new bigraph $A^{'}$. Different from the matching algorithm in [29], the algorithm in this paper takes into considerations the attack-aware information in each agent (bigraph without sites) that is changing as time goes on. The granularity reaches down to the node (instance) level instead of the control (abstract) level, and it uses conditions on the reactions, including logic expressions, time and probabilities. Algorithm 1 shows the bigraph matching algorithm using the term language.

Let R be a set of reaction rules, C_r be the number of controls in each reaction rule, C_a be the number of controls in an agent, and those controls are either nested one by another, or at the same level of the containments; and L be the reaction steps or lengths. The computation complexity of bigraph matching is $O(|R| \cdot M)$, where M represents $C_a \cdot C_r$, and the computation complexity to generate a *B-LTS* is $O(|R| \cdot M \cdot L)$. Assume each reaction rule fires once along the reaction path, the computation complexity of bigraph matching is $O((|R|)^2 \cdot M)$.

B. Architecture of the Proposed Approach

Figure 5 shows the architecture of the proposed approach, which consists of two portions, bigraphical modeller (BigM)



Boolean:	Reaction conditions, eg:'x==2, y==3', separated by commas	Assignment:	Please enter an expression
Input:	Input condition	Output:	Output
Rule name:	cbc01	Probability:	20
Time:	Time for reaction, enter a number	isRandom:	☑

Fig. 3. Modelling attack templates with reaction rules

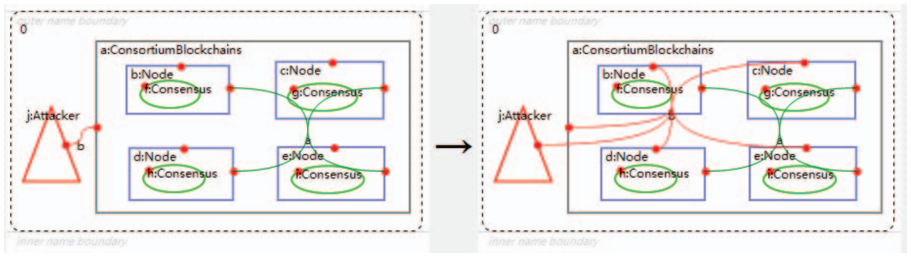


Fig. 4. Modelling attack templates with more reaction rules

of attack graphs as described in the previous section, and bigraphical simulation (BigSIM) to generate attack graphs.

1) Bigraphical modeller, BigM: Setting on top of the architecture, BigM provides three major functions, specifying the signature, modelling agents, and modelling reaction rules.

The signature defines the place sort and link sort related to attack graphs for BC system. For example, the place sort can specify that consensus algorithm is installed in a physical machine, and the machine is sitting inside a safe zone of a validation node participating in block creating, voting, and storing; while the link sort can specify that consensus machines has three ports, corresponding to membership module, smart contract module, and distributed ledger module.

The agent is a bare bigraph, i.e., it has no site, and played as the start state of system configuration. The reaction rules depicts the dynamic behaviours of BC system, especially from the attack point of view. For example, a redex of a reaction represents an attack with user permission, and reactum depicts that the attack obtains the root permission.

In addition, BigM tool allows to model conditions of reaction rules, specify the probabilities of rules fired, and define the attributes of models to be checked. Term language is used as the internal representation of agents and reaction rules.

2) Bigraphical Simulation, BigSIM: It uses rewriting technology to perform matching and deriving. New bigraph states of *B-LTS* will be searched respectively by breadth-first search to find the matching bigraph and implement replacement, in order to complete the derivation process of the system model. BigSIM produces attack graphs automatically, and presents to users in a visual way.

V. Optimization of Generating Attack Graphs

This section first investigates the characteristics of BC system, and reaches a conclusion that the monotonicity assumption cannot be applied in BC system for attack graph generation. Then possible optimization approaches are discussed.

A. Discussion on monotonicity in BC system

Hewett and Kijsanayothin propose a methodology that uses a host-centric modelling approach together with a monotonicity assumption [1] to alleviate the scalability problem of model checkers. The monotonicity assumes that the pre-conditions of an exploit that are once validated remain true. Not even future exploits can invalidate these pre-conditions. As the number of exploits increases, the number of validated pre-conditions increases, a monotone (non-decreasing) behaviour.

However, in a BC system, e.g., with four *v*-nodes, when a *v*-node is ruled out by consensus mechanism, the *v*-node will be carried out a thorough health check, eradicate the pre-conditions of exploits. and get recovered by synchronizing missed blocks from other three normal *v*-nodes, and obtaining

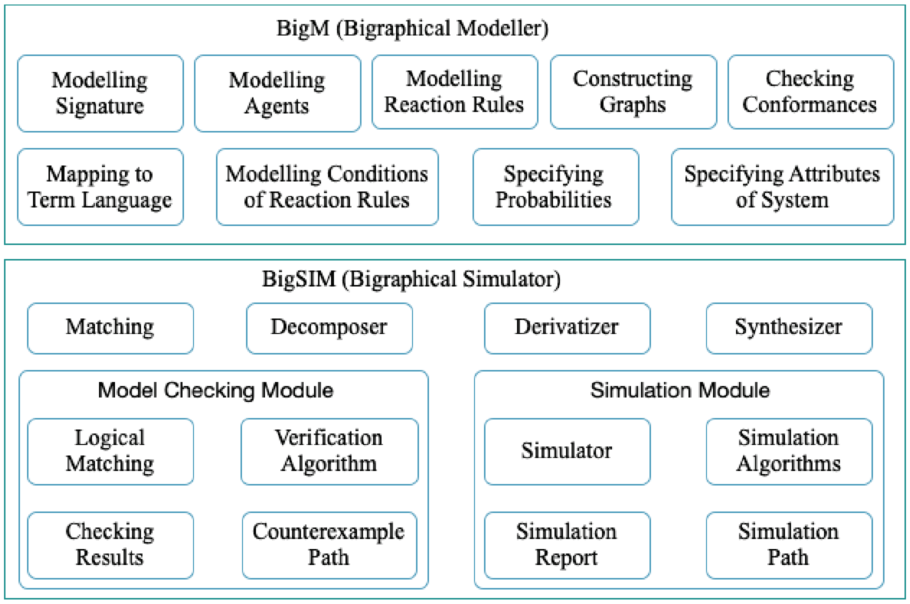


Fig. 5. Architecture of the proposed approach based on bigraph theory

the missed ones from them that sustain the normal behaviours of BC system. After self-healing, the attacked *v*-node joins back to the BC activities.

Figure 6 shows the initial state of BC system, where *v*-node *a* has 99 blocks and missed one block due to a BC attack. Reaction rule in Figure 7 depicts that the health check (H-check) is activated by the BFT consensus result, and reaction rule in Figure 8 illustrates that synch module in *v*-node *a* autonomously takes action to pull the missed block from other three *v*-nodes and renders *v*-node *a* back to normal. The corresponding term language representations of reaction rule in Figure 7 are illustrated as follows:

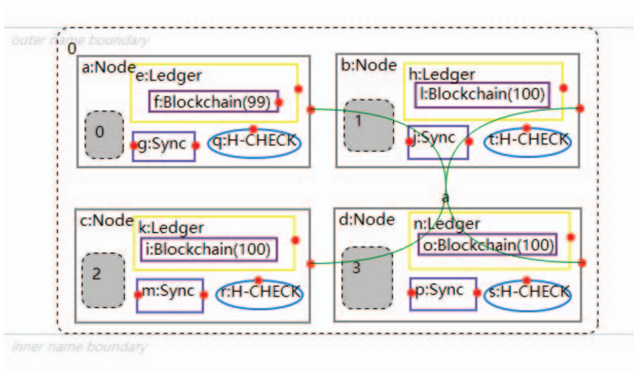


Fig. 6. One *v*-node misses a block due to an attack with some exploit

```
%rule Figure 7 a:Node[a:edge].(e:Ledger[idle].
f:Blockchain_99_[idle] | g:Sync[idle,idle] |
q:H_CHECK[idle] | $0) | b:Node[a:edge].
(h:Ledger[idle].i:Blockchain_100_ |
j:Sync[idle,idle] | s:H_CHECK[idle] | $1) |
c:Node[a:edge].(k:Ledger[idle].l:Blockchain_100_ |
m:Sync[idle,idle] |r:H_CHECK[idle] | $2) |
d:Node[a:edge].(n:Ledger[idle].o:Blockchain_100_ |
p:Sync[idle,idle] | t:H_CHECK[idle] | $3) ->
a:Node[a:edge].(e:Ledger[idle].
f:Blockchain_99_[b:edge] | g:Sync[idle,idle] |
q:H_CHECK[b:edge] | $0) |
b:Node[a:edge].(h:Ledger[idle].i:Blockchain_100_ |
j:Sync[idle,idle] |s:H_CHECK[idle] | $1) |
c:Node[a:edge].(k:Ledger[idle].l:Blockchain_100_ |
m:Sync[idle,idle] | r:H_CHECK[idle] | $2) |
d:Node[a:edge].(n:Ledger[idle].
o:Blockchain_100_ | p:Sync[idle,idle] |
t:H_CHECK[idle] | $3){};
```

As a result, an attacker has to back-track and this invalidates the assumption of monotonicity, thus host-centric approach become unsuitable. In order to handle the scalability problem for generating attack graphs in BC system, this paper explores the following methods.

B. Exploit flow for attack actions

In a BC system, there are a volume of *v*-nodes, and the number of *v*-nodes increases and decreases dynamically. Each *v*-node is a cluster with a plethora of hosts or machines, communication facility, and storages. Each host/machine installs/uninstalls a variety of software, and mounts/dismounts a bunch of hardware. To represent such dynamics of BC system would need a pile of reaction rules, and the number of states in the labelled transition system will increase explosively.

This paper identifies attack exploits to concentrate on attacking-related behaviours regarding to bigraphical reaction rules. Three types of attack exploits are specified, exploit-define, exploit-computation-use, and exploit-predicate-use.

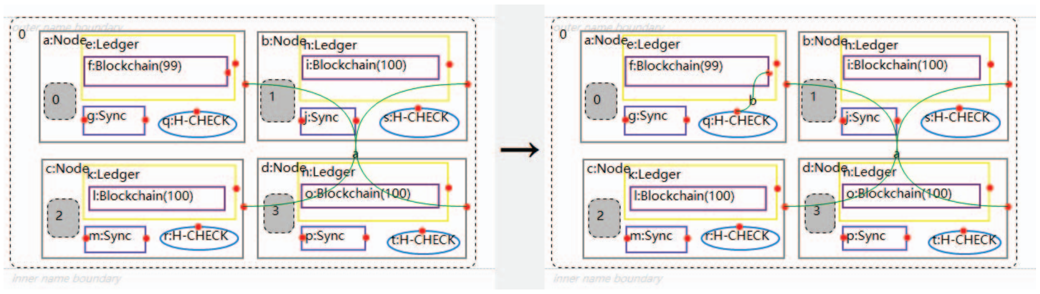


Fig. 7. The synch module of the node with missed block initiates synchronization

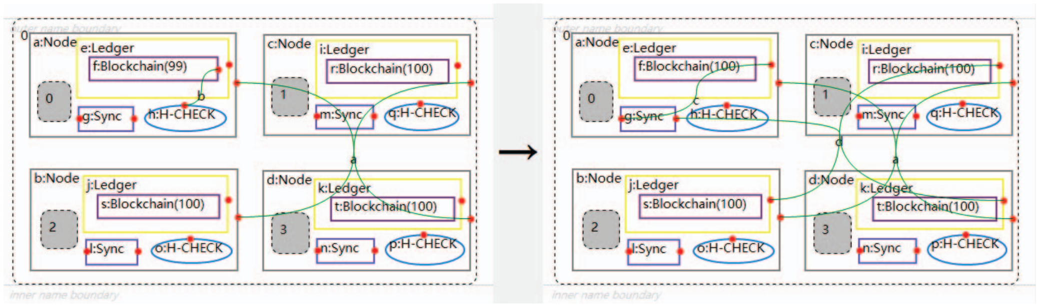


Fig. 8. The missed block is obtained by synch, and BC system gets back to normal

- Exploit-define: An attack exploit structure that appears in reactum and does not appear in redex in a reaction rule.
- Exploit-cuse: An attack exploit structure that appears in redex and does not appear in reactum in a reaction rule.
- Exploit-puse: An attack exploit structure that appears both in redex and reactum in a reaction rule.

Attack exploit flow keeps tracing the attack-define, attack-cuse and attack-puse, and generates attack graph eventually. Two exploit flow-based strategies to generate attack graphs are proposed that select a set of reaction paths such that the set of paths meet the all-exploit-defs or all-exploit-uses criteria. The all-exploit-defs criterion selects a set of reaction paths π in *B-LTS*, such that π covers each definition of each exploit specified, to some use of exploits; and the all-exploit-uses criterion selects a set of reaction paths π, such that π covers each definition of each exploit specified, to each use of exploit.

The exploit-flow approach is different from traditional data flow testing in two aspects, summarized as follows: def-use pairs on the identified paths in the program regarding variables, and execution paths must be known beforehand; on the other hand, def-use pair on the reaction rules regarding a static structure of a sub-bigraph, flows can be identified based on reaction rules without generating execution paths, and simulation can also generate possible execution paths to reveal attack paths.

VI. Experiment and Analysis

A. Experiment setting

The experiment sets up a BC consortium with four *v*-nodes, representing four independent parties without mutual trusting. Each *v*-node has machines to install consensus module, smart contracts, synch module, and distributed ledger. The consortium has a firewall, and each *v*-node has its own firewall. The initial setting is depicted with the bigraph agent in Figure 2. The dynamic aspects of BC system are described with 14 reaction rules, and listed as follows.

1) An attacker attacks successfully the firewall of BC consortium
2) An attacker attacks the firewall of each *v*-node in the consortium
3) An attacker attacks successfully one firewall on one v-node in the consortium
4) An attacker attacks successfully the consensus module on one *v*-node in the consortium
5) One of the attacked nodes in the consortium is abandoned by the consortium
6) The attacker successfully attacks a firewall on the second node in the consortium
7) The attacker successfully attacks the consensus module on the second node in the consortium
8) The attacker successfully attacks the consensus module on a node in the consortium
9) The second node attacked in the consortium is abandoned by the consortium
10) The attacker successfully attacks a firewall on the third node
11) The attacker successfully attacks the consensus module on the third node in the consortium
12) The attacker attacks successfully the consensus module on a *v*-node in the consortium

Algorithm 1 BigraphMatching (A, $\mathcal{R}$)

```
Input: Bigraph $A$ (an initial agent), and a set of reaction rules $\mathcal{R}$;
Output: A set of new bigraphs, $\mathcal{A}$;
1: for each reaction rule $r$ in $\mathcal{R}$ do
2:     flag = findMatch(agent, redex); // redex of r
3:     if (flag==TRUE AND timeConstraints == TRUE AND reactionProbability $\geq$P) then $\mathcal{A}$ = $\mathcal{A} \cup$ { A | replace redex in A with reactum in r }
4: end for
5: return a set of new bigraphs
```

Method:findMatch (agt, red)

```
Input: Term agt from an agent, term red from a redex of a rule;
Output: A flag to indicate whether the match is found;
1: if (agt.termType == PREFIX && red.termType == PREFIX)
2: then matchPrefixPrefix(agt, red) // Parameter agt is a prefix, Parameter red is a prefix; recursively match both the prefixes and suffixes in agt and red; if yes, return TRUE, otherwise return FALSE
3: else if (agt.termType == PARALLER && red.termType == PREFIX)
4: then matchParallerPrefix(agt, red) //Parameter agt is a paraller, Parameter red is a prefix; return FALSE
5: else if (agt.termType == PREFIX && red.termType == PARALLER)
6: then matchPrefixParaller(agt, red) //Parameter agt is a prefix, Parameter red is a paraller; if red has only one non-site sub-node, recursively match the sub-node with agt; if yes, return TRUE, otherwise return FALSE
7: else if (agt.termType == PARALLER && red.termType == PARALLER)
8: then matchParallerParaller(agt, red) // Parameters agt and red are both parallers; recursively match the nodes of agt and red; if yes, return TRUE, otherwise return FALSE;
9: else if (red.termType == TermType.SITE)
10: then matchTermSite(agt, redex) //Parameter agt is a term, Parameter red is a site, return TRUE
11: else if (agt.termType == NIL && red.termType == NIL)
12: then matchNilTerm(agt, red) // Parameter agt is a Nil, Parameter red is a term; if term red is a site or Nil, return TRUE, otherwise return FALSE
13: return a set of new bigraphs
```

13) An attacker attacks successfully the consensus modules on two *v*-nodes in the consortium
14) Two *v*-nodes are attacked, and BC consortium cannot create new blocks

Two tools are developed to support modelling and simulation analysis, BigM and BigSIM. BigM is used to model the agent and the reaction rules, and BigSIM to perform the simulation by matching the configuration with rules, and automatically output labelled transition system to represent attack graphs.

The modelling and simulation is running on Dell machine with CPU Intel (R) Core i5-4210U@1.70GHz, RAM 8GB, installed Windows 10 Pro 64bit.

TABLE I
COMPARISON WITH AND WITHOUT EXPLOIT FLOW

Compare	Reaction rules	States	Time (second)
Exploit flow	14 related	12	1-1.5
Not applying	40 in total	32	4-5

B. Results and analysis

By applying the matching algorithm described in Section IV-A, BigSIM generates the labelled transition system, i.e., attack graph, where the terminal nodes represent successful attack by attackers, as shown in Figure 9. The attack graph starts from the agent bigraph, and ends up in three states, F, I, and L, corresponding to three bigraphs, percentage numbers on the arrows indicate the probabilities that the attacks can success, and the numbers within the states of F, I, L, indicate the largest joint probabilities of successful attacks from possible paths.

Bigraph F indicates that the consensus module on only one *v*-node gets attacked, and the *v*-node cannot participate in voting, while the BC system works as normal, i.e., blocks are produced as usual. Bigraph I indicates that the consensus modules on two *v*-nodes get attacked, and make the BC system unable to reach consistent results, i.e., blocks cannot be produced any more. Bigraph L indicates the worst case where the attacker seizes the BC system with three *v*-nodes under its control, and can do whatever he wants to do. Compared with bigraphs I and L, bigraph F has higher probability to reach, up to 0.0066%, against 0.005% and 0.0048%. That is there is higher probability for BC system to work normally with fault tolerance capability.

When adding more reaction rules to BC system, BRS will match these rules and derive more bigraph states, leading to increasing of computation time. Figure 10 shows another attack graph generated by BigSIM automatically, where there are 40 reaction rules in total, and does not apply exploit flow approach. AE state indicates that three *v*-nodes are successfully attacked, AB state two *v*-nodes and BC state only one *v*-node. To reduce the computation effort, this paper applies exploit flow approach to prune the unrelated branches that have no-possibility to be attacked. Table I shows the comparison between approaches with and without exploit flow, where the former generates 32 states vs. 12 states in the later, and taking 4 to 5 seconds vs. 1 to 1.5 seconds.

VII. RELATED WORK

The attack graph is one of the security assessment models for modelling cyber attacks. It was first proposed by Phillip and Swiler [34] in the 1990s. The attack graph is a directed

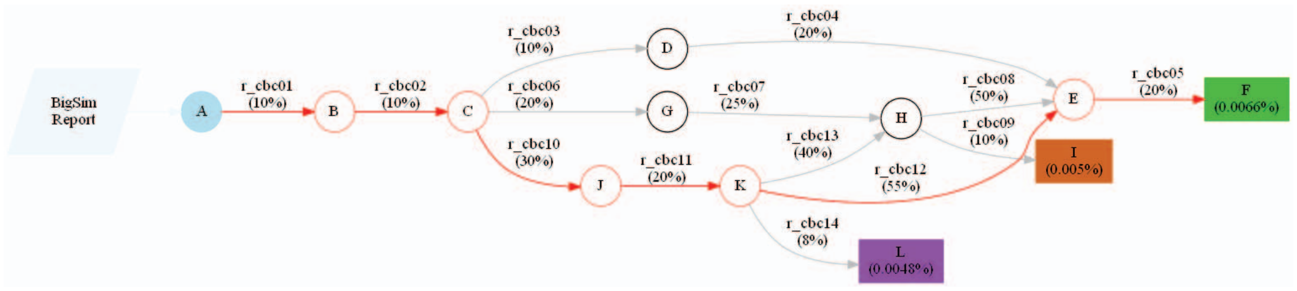


Fig. 9. Generating attack graph based on bigraph theory with exploit flow approach

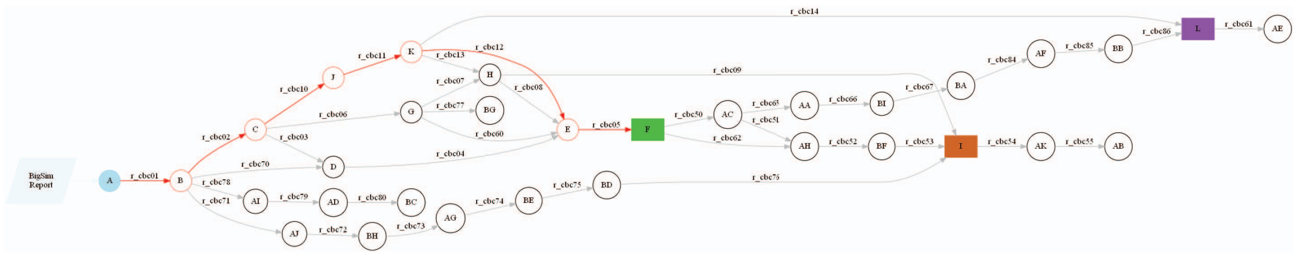


Fig. 10. Generating attack graph without applying exploit flow approach

graph consisting of vertices and directed edges which describes the attack sequence and attack effect that an attacker may launch.

At present, there are three main methods for generating attack graphs: the first one is the method using model checking. In order to obtain an attack scenario, such a model must contain all the network states to make the model too complex, and the generated attack graph is very large. Ritchcy [31] et al. proposed this method for the first time. The elements in the network are first encoded, including descriptions of the host, reachability between hosts, the attacker's starting point, and the attacks available to the attacker. After the modelling is completed, a finite state machine model is used to represent the change of the network state, and then the detection tool SMV is used to find out whether there is a violation of the security setting in the network, that is, a counterexample. But the model is too simple to represent some complex attacks. Sheyner [32] et al. improved the model in 2002, using the NuSMV developed by SMV to find the counterexamples in the network, and gave all the paths to reach this counterexample, so that the attack graph can be used to further analyse the network environment.

The second method is to generate an attack graph based on the idea of graph theory. Ammann [1] et al. proposed a graph-based search method in 2002, and pointed out that there is a scalability problem in the generation method of model detection. It is difficult to directly generate an attack graph for a relatively simple network. Feng Huiping [13] et al. proposed a vulnerability analysis model based on reliability theory, and used the model detection iterative algorithm to search the state set technology to automatically construct the attack graph. Compared with the model checking method, this method has relatively good spatial complexity and time complexity. However, the attack template required to generate an attack scenario cannot be flexibly added or modified, and the scale of the attack graph is not solved.

The third method is proposed by Bhattacharya [5] and Ghosh, who use intelligent planning techniques to automatically build state diagrams. Since the traditional method of constructing an attack graph requires a comprehensive analysis of the interaction between the target network and the state of the attacker, there is inevitably a problem of state explosion, resulting in an excessively large attack graph, which greatly reduces the practicality of attack graph. In order to solve this problem, the researchers have proposed some ways to solve this problem. Noel [30] [17] [24] [25] [26] et al. used the method of hierarchical merging and adjacency matrix to reduce the number of nodes in the attack graph, making the generation of the attack graph easier to visualize, and not fundamentally solving the problem of the scale of the attack graph. Li [18] et al. proposed a forward search attack graph generation algorithm based on hypergraph partitioning. Before the attack graph is generated, the host with the same characteristics is merged and the attack graph is generated from the target node to the attacker to reduce the storage attacker. The additional resources required by the current state effectively reduce the number of edges and nodes in the attack graph, further reducing the scale of the attack graph, and improving the attack graph generation efficiency of large-scale complex networks. But it can only be applied to vulnerability [19] analysis of some special networks. Dawkins [7] uses the strategy of limiting the attack step to generate the attack tree based on the breadth-first search algorithm. However, because the breadth-first search algorithm is adopted in his proposed method, there may be a situation that when the maximum attack step is reached, the target state is still not reached, resulting in a large number of non-target state leaf nodes in the generated attack tree.

Ammann [1] et al. first proposed the monotonic hypothesis of the attacker's ability, that is, the attacker's ability to attack during the attack is constantly increasing, and the ability that has been acquired in each atomic attack will not be lost. The complexity of attack graph construction is reduced from exponential complexity to polynomial complexity. Ou [2] [27] et al. proposed a method based on logic programming technology to generate logical attack graphs. The specific method is to propose a logic-based network security analyser MulVAL, which can carry out on more hosts, multi-level network security analysis and reasoning further reduce the complexity of generating attack graphs, and experimentally verify that the method can generate 1000 host-scale network attack graphs within 20 minutes. Based on the AGML modelling language, Chen Feng [10] uses attack mode filtering technology, attribute compression technology and instantiation checking technology to pre-process attack exploits and attributes, thus reducing the complexity of the algorithm. Ye Yun [37] et al. adopted the AGML modeling language proposed in [10] to reduce the number of matches and reduce the complexity of the algorithm by pre-classifying the target environment attributes according to the host and predicates.

Fan Zihua [38] et al. proposed a Rete-based attack graph construction method, which has good construction efficiency and can be used to construct attack graphs in large-scale networks. Moulin [22] et al. describe a way to automatically build attack graphs in the network by using time logic. Qing Dapeng [19] et al. proposed a depth-first attack graph generation method. The depth-first search algorithm was used to find the attack path in the network. The strategy of limiting the number of attack steps and the success probability of the attack path was used to reduce the scale of the attack graph and pass the experiment. Experiments show that the method can effectively remove redundant edges and nodes in the attack graph, thus reducing the scale of the attack graph.

Shi Hao [33] proposed the attack graph generation method based on expert knowledge. This method stands on the perspective of the network administrator and assumes that the network topology is known, so that the attack graph can be generated from the key nodes of the network. The generated attack graph is simple and effective, rarely has unnecessary redundant information, and is useful for analysing the generated attack graph. Qin Hu [15] et al. proposed an attack graph generation method based on the privilege lifting matrix. The matrix describes the attacker's privilege escalation process during the attack process, which reduces the algorithm complexity of generating the attack graph and is beneficial to the attack graph generation in large-scale network environment.

While bigraph has been extensively developed in the last 20 years, yet significant work is still needed to address large problems. Perrone [28] provides a sorting logic [11] that always produces well-behaved sorting predicates, and this paper uses the logic as a mechanism to constraint place graphs and link graphs. However, the completeness of the sorting logic needs to be investigated in the future. Bigraphs are created to model the behaviours of large mobile systems [36], context-aware system [35] [6], and information processing applications [14], but have not been applied to generate attack graphs yet.

Faithful [12] developed a software tool, BigRed, for editing the signature, bigraphical agent, and BRS rules in the term languages. The tool does not support modelling the sorting information, and does not have data elements in its model. BigM [39] is a new tool by extending BigRed with more advanced features, adding sorting information to allow users to allocate place sort to each control and link sort to each port; realizing the sorting logic user interface by adding constraints in the bigraphical models; developing a data model to store data, using bigraphical signature to specify its class, attributes and instances; supporting to specify the guard conditions and probability specification on the reaction rules.

Another important software tool to support the applications is BigSIM [39], a bigraphical simulator implemented using Scala [40], which extends BigMC, a bigraphical model checker, developed by Gian [29] using C++. BigMC does not support sorting logic, timing specification, relational expressions and guard conditions on reaction rules. BigSIM evolves BRSs based on current agents and reaction rules represented by a term language to simulate the real world transitions.

VIII. Conclusion

This paper explores the characteristics of BC system, where tolerance and self-healing mechanisms are applied, and thus the monotonicity assumption is not suitable for the attack graph generation. Although model checking approach is well applied in automatic generation of attack graphs, current model checking tools do not provide a comprehensive approach to model the three aspects of attack graphs, i.e., attack templates, configuration and attacker profiles. This paper proposes to apply bigraph theory to attack graph generation. Place sort and link sort are used to model configuration of BC system and attacker profile, and bigraphical reaction rules to model attack templates and attacker behaviours. Bigraphical matching algorithm derives labelled transition system to automatically generate attack graphs. To mitigate matching complexity, this paper proposes to apply exploit flow mechanism to accelerate the attack graph generation by concentrating on interested exploit pattern related to attacks. Preliminary results of the experiment show the exploit flow approach shows good performance in terms of time consumption. More experiments and evaluations will be carried out in the future work.

Acknowledgement

The authors would thank Xin Li, Yacong Dong, Ziying Chen and Anqiang Shen for their helps with running experiments and processing features in this research. The authors would

also thank the anonymous reviewers for their invaluable comments.

REFERENCES

[1] P. Ammann, D. Wijesekera, S. Kaushik, "Scalable graph-based network vulnerability analysis," Proceedings of the 9th ACM Conference on Computer and Communications Security. pp. 217-224. ACM, 2002.

[2] A.W. Appel, S. Govindavajhala, X. Ou, "Mulval: A logic-based network security analyzer," In: USENIX security symposium. vol. 8. Baltimore, MD, 2005.

[3] F. Baader, & T. Nipkow, *Term Rewriting and All That*. Cambridge: Cambridge University Press. 1998.

[4] M.S. Barik, A. Sengupta, C.M, "Attack graph generation and analysis techniques," pp. 559-567, 2016.

[5] S. Bhattacharya, S. Ghosh, "An articial intelligence based approach for risk management using attack graph," 2007 International Conference on Computational Intelligence and Security (CIS 2007). pp. 794-798. IEEE, 2007.

[6] L. Birkedal, S. Debois, E. Elsborg, T. Hildebrandt, H. Niss, "Bigraphical models of context-aware systems,". International Conference on Foundations of Software Science and Computation Structures. pp. 187-201. Springer , 2006.

[7] H.K. Browne, W.A. Arbaugh, J. McHugh, W.L. Fithen, "A trend analysis of exploitations," In: Proceedings 2001 IEEE Symposium on Security and Privacy. S & P 2001. pp. 214-229. IEEE, 2001.

[8] J. Cabot, M. Gogolla, "Object Constraint Language (OCL): A Definitive Guide", The 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18-23, 2012. Advanced Lectures (pp.58-90).

[9] M. Castro, B. Liskov, "Practical Byzantine Fault Tolerance", in the Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, USA, February 1999.

[10] F. Chen, "A hierarchical network security risk evaluation approach based on multi-goal attack graph," National university of defense technology, Chang sha, 2009.

[11] S. Debois, "Sortings and Bigraphs. Ph.D. thesis," IT University of Copenhagen, 2008.

[12] A.J. Faithfull, G. Perrone, T.T. Hildebrandt, "Big red: A development environment for bigraphs," Electronic Communications of the EASST 61, 2013.

[13] P.H. Feng, Y.F. Lian, Y.X. Dai, X.H. Bao, "Vulnerability model of distributed systems based on reliability theory," Ruan Jian Xue Bao(Journal of Software) 17(7), 1633-1640, 2006.

[14] T. Hildebrandt, H. Niss, M. Olsen, "Formalising business process execution with bigraphs and reactive xml," In: International Conference on Coordination Languages and Models. pp. 113-129. Springer, 2006.

[15] Q. Hu, J.L. Wang, P.X, "Attack graph generation method based on privilege escalation matrix," pp. 101-105, 2019.

[16] S. Jajodia, S. Noel, "Topological Vulnerability Analysis", Cyber situational awareness. Issues and research, pp.139-154.

[17] S. Jajodia, S. Noel, B. O'berry, "Topological analysis of network attack vulnerability," In: Managing Cyber Threats, pp. 247-266. Springer, 2005.

[18] H. Li, Y. Wang, Y. Cao, "Searching forward complete attack graph generation algorithm based on hypergraph partitioning," Procedia Computer Science 107, 27-38, 2017.

[19] D.P. Man, B. Zhang, Y. Zhou, W. Yang, Y.T. Yang, "Depth-first method for attack graph generation [j]," Journal of Jilin University (Engineering and Technology Edition) 2, 2009.

[20] R. Milner, "The space and motion of communicating agents," Cambridge University Press, 2009.

[21] C. Modi, D. Patel, P. Swathi, "Preventing Sybil Attack in Blockchain using Distributed Behavior Monitoring of Miners," 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 2019, pp. 1-6, doi: 10.1109/ICCCNT45670.2019.8944507.

[22] M. Moulin, E. Eyisi, D.M. Shila, Q. Zhang, "Automatic construction of attack graphs in cyber physical systems using temporal logic," In: MILCOM 2018 IEEE Military Communications Conference (MILCOM). pp. 933-938. IEEE, 2018.

[23] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf, 2008.

[24] S. Noel, M. Jacobs, P. Kalapa, S. Jajodia, "Multiple coordinated views for network attack graphs," In: IEEE Workshop on Visualization for Computer Security, 2005.(VizSEC 05). pp. 99-106. IEEE, 2005.

[25] S. Noel, S. Jajodia, "Managing attack graph complexity through visual hierarchical aggregation," In: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security. pp. 109-118. ACM, 2004.

[26] S. Noel, S. Jajodia, B. O'Berry, M. Jacobs, "Efficient minimum-cost network hardening via exploit dependency graphs," In: 19th Annual Computer Security Applications Conference, 2003. Proceedings. pp. 86-95. IEEE, 2003.

[27] X. Ou, W.F. Boyer, M.A. McQueen,, "A scalable approach to attack graph generation," In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 336-345. ACM, 2006.

[28] G. Perrone, "Domain-specific modelling languages in bigraphs," Ph.D. thesis, IT University of Copenhagen, Programming, Logic and Semantics, 2013.

[29] G. Perrone, S. Debois, T.T. Hildebrandt, "A model checker for bigraphs," In: Proceedings of the 27th Annual ACM Symposium on Applied Computing. pp. 1320-1325. ACM, 2012.

[30] R. Ritchey, B. O'Berry, S. Noel, "Representing tcp/ip connectivity for topological analysis of network security," In: 18th Annual Computer Security Applications Conference, 2002. Proceedings. pp. 25-31. IEEE, 2002.

[31] R. Ritchey, R.W, P. Ammann, "Using model checking to analyze network vulnerabilities," In: Proceeding 2000 IEEE Symposium on Security and Privacy. S& amp;P 2000. pp. 156-165. IEEE, 2000.

[32] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J.M. Wing, "Automated generation and analysis of attack graphs," ln: Proceedings 2002 IEEE Symposium on Security and Privacy. pp. 273-284. IEEE, 2002.

[33] H. Shi, Y.J. Wang, Z. Xue, "Attack graph generation based on security administrator information," Information Security & Communications Privacy, 2011.

[34] L.P. Swiler, C. Phillips, "A graph-based system for network-vulnerability analysis. Tech. rep., Sandia National Labs., Albuquerque, NM (United States), 1998.

[35] L. Yu, W.T. Tsai, G. Perrone, "Testing context-aware applications based on bigraphical modeling," IEEE Transactions on Reliability 65(3), 1584-1611, 2016.

[36] L. Yu, W.T. Tsai, X. Wei, J. Gao, X.Q. Guo, T.T. Hildebrandt, "Modeling and analysis of mobile cloud computing based on bigraph theory," In: 2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering. pp. 67-76. IEEE, 2014.

[37] Y. Yun, X. X. Shan, W. X. Yang, Q. Z. Chang, "Attack graph generation algorithm for large-scale network system," Journal of Computer Research and Development 50(10), 2133-2139, 2013.

[38] F. Zihua, "Generation method of attack graph based on rete algorithm," Compuer Engineering 44(3), 151 (2018), http://www.ecice06.com/EN/abstract/article_28565.html.

[39] https://github.com/LongyunZhang/PKU_BigM_BigSIM.

[40] The Scala Programming Language: https://scala-lang.org/.

[41] Selfish mining attack: https://golden.com/wiki/Selfish_mining_attack.

[42] Mining malware: https://www.cryptowisser.com/crypto-mining-malware/.

[43] 51% attack: https://www.investopedia.com/terms/1/51-attack.asp.

[44] Timejack attack: https://blogs.arubanetworks.com/solutions/10-blockchain-and-new-age-security-attacks-you-should-know/.

[45] Finney attack: https://bitcoin.stackexchange.com/questions/4942/what-is-a-finney-attack.

[46] Race attack: https://bitcoin.stackexchange.com/questions/60164/race-attack-vs-double-spending-attack-are-they-same.

[47] The DAO attack: https://www.coindesk.com/understanding-dao-hack-journalists.

[48] Parity Multisig Wallet Attack: https://www.parity.io/parity-technologies-multi-sig-wallet-issue-update/.

[49] Eclipse Attacks on Blockchains'Peer-to-Peer Network: https://hackernoon.com/eclipse-attacks-on-blockchains-peer-to-peer-network-26a62f85f11.