Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 8812017, 11 pages
https://doi.org/10.1155/2020/8812017

Research Article

An Invocation Chain Test and Evaluation Method for Fog Computing

Yue Zhao,[1,2,3] Yarang Yang,[4] Bo Tian,[1,2,3] and Tianyi Zhang[5]

[1]*Science and Technology on Communication Security Laboratory, Chengdu 610041, China*
[2]*No.30 Research Institute of China Electronics Technology Group Corporation, Chengdu 610041, China*
[3]*China Electronics Technology Cyber Security Co., Ltd., Chengdu 610041, China*
[4]*College of Physics and Electrical Engineering, Kashi University, Kashi 844006, China*
[5]*Graduate School of Advanced Integration Science, Chiba University, Chiba 2638522, Japan*

Correspondence should be addressed to Yarang Yang; sxyyr@163.com

Received 23 April 2020; Revised 18 June 2020; Accepted 7 July 2020; Published 28 August 2020

Academic Editor: Fuhong Lin

In this paper, an invocation chain technology is used to test the security of fog computing systems. Atomic attacks based on the attack graph are combined according to the type, time sequence, and causal relationship. Different test sequences have gone through according to the principle of depth-first. In addition, the vulnerability assessment based on an invocation chain is evaluated to verify whether it can detect existing or unknown vulnerability in fog computing systems. Finally, the experimental results show that the invocation chain test and evaluation method based on the attack graph can evaluate the system risk quantitatively rather than qualitatively by calculating comprehensive probability on all test sequences.

1. Introduction

Fog computing extends cloud computing services to the edge of the network. There are many security threats in network architecture of cloud computing that have been analysed in [1, 2]. Under the distributed architecture of fog computing, there are numerous fog nodes and complicated connections. The security threats faced by the whole fog computing network system are quite different from those of the traditional cloud computing system. The vulnerabilities are often referred to as computer hardware, software, or policy flaws that allow attackers to access or destroy a system without authorization. There are many known and unknown vulnerabilities in the fog nodes, and there are problems that the security updates of the fog nodes and the central cloud is not synchronized in time. Once it is broken, the user data connected to the insecure fog nodes are faced with a variety of security threats [3]. In the scenarios of enterprise networks and Internet of Things, the fog nodes are geographically dispersed and exposed, and they are vulnerable to attack at the hardware level. Since fog computing systems have the difference of device structure, protocol, and service provider, the existing intrusion detection technology has difficulty detecting the above attacks [4].

Vulnerabilities exist in all aspects of design, implementation, and operations management of the fog computing system. It is unrealistic to eliminate all vulnerabilities in the system. It is well proved that the impact of a single vulnerability among others is quite small, while the impact of the vulnerabilities becomes significant when they are interlinked. Many examples prove that while the impact of several isolated vulnerabilities is small, more often than not, the vulnerabilities are interlinked. If the attacker organizes and utilizes these vulnerabilities organically, they will bring great harm to the system. Recently, Bozzato discovered three vulnerabilities in the Samsung SmartThings Hub controller, namely, CVE-2018-3879, CVE-2018-3880, and CVE-2018-3917 [5]. The attacker can get sensitive information leaked from memory in the hub core process by combining these vulnerabilities together, while using any single one of them cannot cause

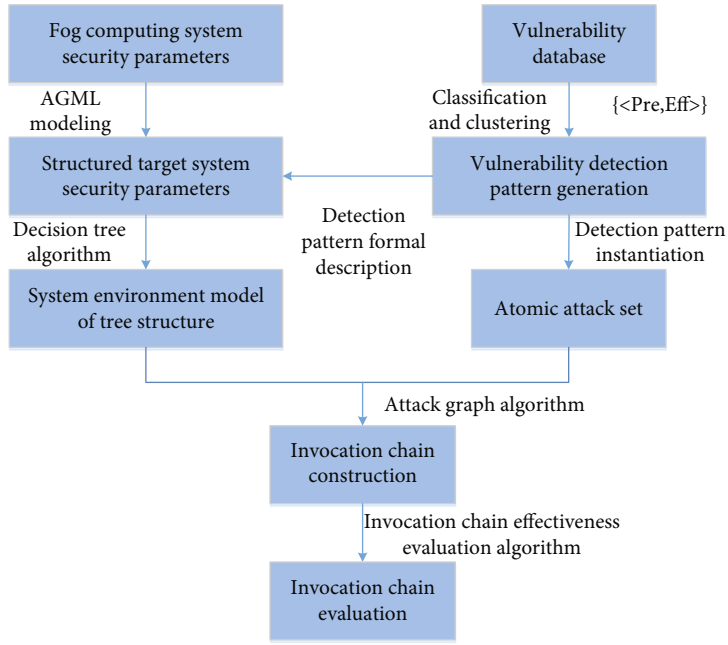

Figure 1: The security test model of fog computing.

the smart home device to suffer network attacks. The attacks on computer systems based on vulnerability combination will cause serious harm to the whole system.

Vulnerability assessment is a proactive defence network security technology. Its working principle is to collect fog computing information; find out the entry point where attacks can be implemented; and perform simulated attacks to discover important vulnerabilities of the system. With the diversification and complication of vulnerabilities, the traditional rule-based vulnerability assessment methods have been unable to comprehensively assess the potential threats arising from the interaction of multiple vulnerabilities. A model-based vulnerability assessment method is proposed to analyse the vulnerabilities of the fog computing system from the perspective of the attacker, enumerate all the invocation chains that utilize the combination of the vulnerabilities, and analyse the test effects of different invocation chains one by one [6]. Since the length of the invocation chain grows exponentially with the increase in the host size and number of vulnerabilities of the fog computing system, this approach obviously cannot be applied to large-scale systems. In order to solve the combinatorial explosion problem caused by the model-based vulnerability assessment method, an attribute-based vulnerability assessment method is provided in [7], proposing the attack will be effectively executed when all the preconditions of the attack are satisfied. It clarifies the precondition of each attack and reflects the time sequence and causal relationship between the vulnerabilities used in different invocation chains, thereby making the constructed invocation chain more concise and more effective. However, [7] only proposes the construction framework of the invocation chains and does not elaborate on the specific method of constructing the invocation chains.

The attack graph is a graph theory method in mathematical modelling, which can visually reflect the detailed process of the system being detected. This paper proposes a vulnerability combination evaluation method based on the attack graph. It includes three parts, i.e., (1) to model fog computing system environment; (2) to generate the invocation chains on the basis of the attack graph, which in nature is vulnerability combination, and compute the chance of different invocation chains successfully implementing detection of the system according to the depth-first principle; and (3) to evaluate the performance of the invocation chains to verify whether the invocation chains can detect known or unknown vulnerabilities. The vulnerability combination assessment method based on attack graph proposed in this paper is helpful to carry out deep security detection of the system and provide technical support for improving the antiattack ability of the fog computing system.

2. Threat Modelling for Fog Computing System Environment

As stated above, the open structure and distribution are the characteristics of fog computing, which make it vulnerable and very weak to security threats. The security test model of fog computing is shown in Figure 1. The tester detects the fog computing system environment and obtains the system running status as well as vulnerability information to generate the vulnerability detection mode and build the system environment model. Therefore, the invocation chains are generated by using the attack graph method, and the effectiveness of invocation chains are evaluated to verify the vulnerability combination evaluation method based on the attack graph.

2.1. Acquisition of System Security Parameters. For fog computing system environment modelling, it is very important to find out the initial vulnerabilities and service information

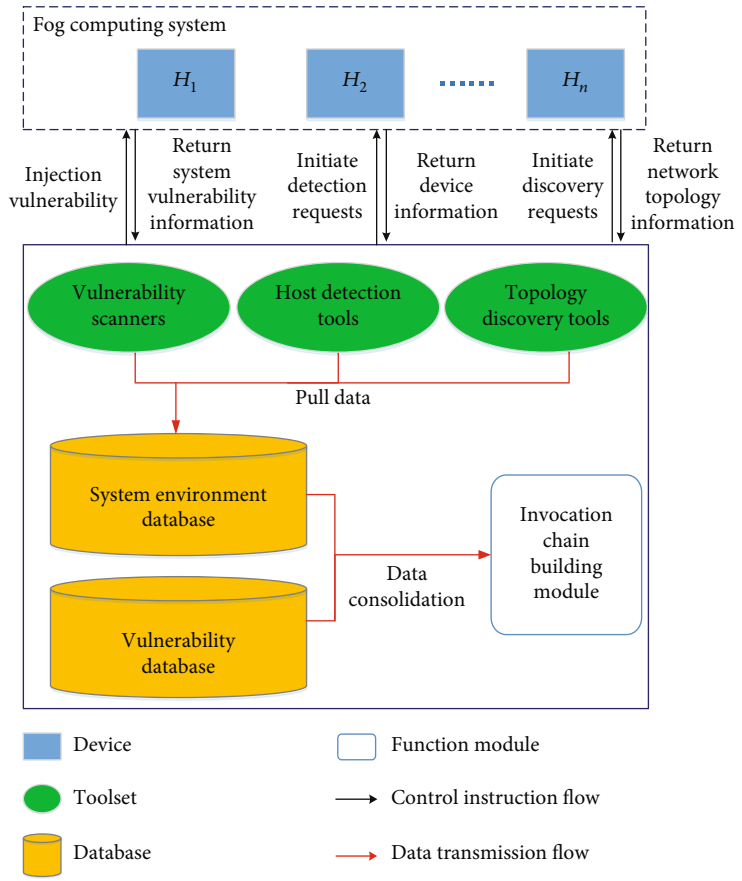

Figure 2: The working mechanism of fog computing system detection.

running on the system through the fog computing system detection toolset. The working mechanism of fog computing system detection is shown in Figure 2. The fog computing system detection toolset includes vulnerability scanners which are designed to discover the vulnerability information of the fog computing system, host detection tools which are able to obtain the configuration information of each host in the system and the applications running on the hosts, and topology discovery tools which can obtain the topology structure of the fog computing system by calculating the connection information among hosts in the network transport layer. The vulnerability information of the fog computing system, the configuration information running on each host, and the connection information among hosts constitute the security parameters of the fog computing system, which are the core elements of fog computing system environment modelling. This paper presents a formal description of the security parameters of the fog computing system and their interactions based on the attack graph modelling language (AGML) [8, 9].

2.2. Security Risk Propagation Model of Fog Computing. Malwares can spread rapidly in fog computing environment and even infect the whole network. Therefore, how to find the hidden danger of each node quickly and reasonably in the fog computing system and avoid the further attacks of

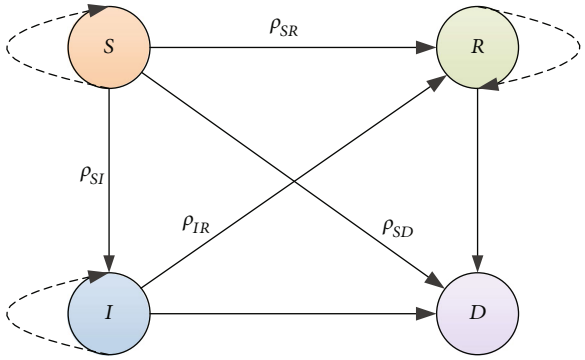

Figure 3: The conversion among these node states.

malicious malwares is an urgent problem to be solved in the fog computing security.

It is assumed that the fog computing system contains N nodes, and these nodes present four different states as follows: susceptible, infectious, recovered, and dead. As the propagation of network attacks is a dynamic process, the state of node varies constantly. Figure 3 shows the conversion among these node states. Supposing all nodes are distributed in the network with density δ, as to a node with state I, the number of its adjacent nodes is $\delta\pi r^2$. It should be noted that only the nodes with state S may be infected due to their vulnerability. The number of communication links formed by

Table 1: Detection pattern of acquisition root access permissions adopting remote memory overflow.

Name	*Re_bof*
Class	α
Vuls	CVE-2003-0245, CVE-2006-2372, CVE-2006-6424, CVE-2006-0026, CVE-2006-2451
Var	*s*: HostID, *d*: HostID, *sn*: ServiceName, *cveid*: CveID, *pro*: Protocol, *port*: Port
Pre	NetworkService(*d*, *sn*, *pro*, *port*), VulExists(*d*, *sn*, *cveid*), Connection(*s*, *d*, *pro*, *port*), RunCode(*s*, *ROOT*)
Eff	RunCode(*d*, *ROOT*), DoS(*d*, *sn*)

the nodes with state S and state I is $\delta\pi r^2 S(t)n_I(t)$. The number of nodes with state S converting into ones with state I is $\mu_{SI}(t)\rho_{SI}\delta\pi r^2 S(t)n_I(t)$ per unit time. ρ_{SI} is the probability that the node with state S and the node with state I are attacked by the infected node in the communication process, and $\mu_{SI}(t)$ is the conversion rate [10]. Meanwhile, the numbers of nodes converting from state S to state R and converting from state S to state D are $\rho_{SR}n_S(t)$ and $\rho_{SD}n_S(t)$, respectively. ρ_{SR} is the probability of installing the vulnerability patches on the nodes with state S and getting them immunized; and ρ_{SD} is the probability of nodes with state S not working properly [11].

Next is the case of installing vulnerability patches on nodes. Assume that there are $\gamma_0 N$ vulnerability patches, where $0 \le \gamma_0 \le 1$. The installation speed of vulnerability patch in the whole fog computing system is $\rho_r\gamma_0 N\mu_R(t)$, where ρ_r depends on the node density, network speed, and other factors in the fog computing system, and $\mu_R(t)$ represents the consumption of network resources by installing vulnerability patches, i.e., $0 \le \mu_R(t) \le 1$. If the node with state S has vulnerability patches installed, it will converse into a node with state R with a certain probability. The number of nodes converting from state S to state R is $\rho_{SR}\rho_r\gamma_0\mu_R(t)\gamma_0 n_S(t)$ per unit time. When the node with state I obtains the vulnerability patch, the probability of it converting to a node with state R, ρ_{IR}, satisfies $0 \le \rho_{IR} \le \rho_{SR} \le 1$. Therefore, the number of nodes converting from state I to state R is $\rho_{IR}\rho_r\gamma_0\mu_R(t)\gamma_0 n_I(t)$ per unit time.

2.3. Generation of Vulnerability Detection Patterns. Vulnerability detection patterns are the methods of finding out vulnerabilities that exist in the fog computing system. They are not used to deal with a single vulnerability but for a certain type of vulnerability. For example, the detailed execution of the buffer overflow in the simple mail transfer protocol (SMTP) is different from the buffer overflow in the hypertext transfer protocol (HTTP). However, if the preconditions and effects of the two tests are the same, the two detection patterns can be classified into the same detection pattern. In this paper, the common attack pattern enumeration and classification (CAPEC) is used to classify the detection patterns [12] and extract the detection patterns with similar preconditions and effects. According to the fog computing system environment, the detection patterns are combined into the corresponding invocation chains, which form the attack graph to test the security of the system.

The descriptive grammar of detection patterns can be formally expressed as triple <*Type*, *Predicate*, *DetectionPattern*>, in which *Type* represents the object type in the detection pattern, *Predicate* represents the predicates of preconditions and effects in the detection pattern, and *DetectionPattern* is described as <*Name*, *Class*, *Vuls*, *Var*, *Pre*, *Eff*>. *Name* is the name of the detection pattern, and *Class* is the type of it. If the *Class* value is α, it is expressed as a vulnerability detection pattern, and if it is β, it is expressed as a vulnerability-independent detection pattern. *Vuls* contains all the vulnerabilities that the detection pattern may exploit. If the *Class* value is β, then *Vuls* is Φ. *Var* denotes the set of variables and their types in the detection pattern. Each element in the set is denoted by $<v : t>$, where v is a variable and t is the type to which the variable belongs. *Pre* and *Eff* are the set of preconditions and the set of effects, respectively, for a certain pattern corresponding to *Predicate*. The logical relationship between the elements in the set is parallel relationship.

For example, Table 1 describes the detection pattern of acquisition root access permissions adopting remote memory overflow, where *Re_bof* is the name of the detection pattern and *Class* is α, indicating that the detection pattern is vulnerability-related. *Vuls* describes that the detection mode is suitable for vulnerability numbers CVE-2003-0245, CVE-2006-2372, CVE-2006-6424, CVE-2006-0026, and CVE-2006-2451 [13, 14]. *Var* indicates all local variables of the pattern and the types of these variables. *s* denotes the source host of the vulnerability detection, *d* is the target host of the vulnerability detection, *sn* is the software applications running on the target host, *cveid* is the vulnerability identification of the software application, *port* is the port of network connection listening, and *pro* is the protocol adopted for network connection. *Pre* represents the preconditions used for this detection pattern: (1) the running service on target host *d* is *sn*, which is provided by communication protocol *pro*, and its listening port is *port*; (2) service *sn* on target host *d* has vulnerability *cveid*; (3) source host *s* can access to port *port* of target host *d* through communication protocol *pro*; (4) it can run service *sn* on source host *s* with the authority root. *Eff* illustrates the two effects of successful application of this pattern: one can run services with privilege root on host *d*; another is service *sn* on host *d* provides denial of service.

2.4. Modelling of Fog Computing System Environment. Based on the decision tree algorithm, the fog computing system environment is modelled with tree structure. First step is to classify hosts; that is, the indexes are built according to the address of hosts; next, classify the type of attribute according to predicate name; at last, classify the type of predicate

Procedure *Pre-Treatment*(*Tree*, *f*)
Input:
Tree of attributes in the system environment;
Attribute *f* in ***Initial*** of System Environment Set
Output:
Tree after inserting attribute *f*
1. $I = IPValue(f)$; //Extracting attribute f of parameter value of *HostID*
2. $P = Predicate\,Name(f)$;
//Extract the predicate name representing attribute *f*
3. **If** $Node(I)$ **in** *S-Nodes*(*Tree.Root*) **Then**
//*S-Nodes*(*Tree.Root*) represents the successive node sets of *Tree*
4. $S\text{-}Nodes(Tree.Root) = S\text{-}Nodes(Tree.Root) \cup \{Node(I)\}$;
5. $S\text{-}Nodes(Node(I)) = S\text{-}Nodes(Node(I)) \cup \{Node(P)\}$;
6. $S\text{-}Nodes(Node(P)) = \{Node(f)\}$;
7. **Else If** $Node(P)$ **in** $S\text{-}Nodes(Node(I))$ **Then**
8. $S\text{-}Nodes(Node(I)) = S\text{-}Nodes(Node(I)) \cup \{Node(P)\}$;
9. $S\text{-}Nodes(Node(P)) = \{Node(f)\}$;
10. **Else If** $Node(f)$ **in** $S\text{-}Nodes(Node(P))$ **Then**
11. $S\text{-}Nodes(Node(P)) = S\text{-}Nodes(Node(P)) \cup \{Node(f)\}$;
12. **Return** *Tree*

Algorithm 1: Environmental modelling for fog computing.

according to its attribute. This decision tree-based classification algorithm can reduce many unnecessary search and matching processes and speed up the process of system environment modelling.

The method of fog computing environment modelling is shown in Algorithm 1. The input of the algorithm is *Tree* of system environment attributes and attribute f of system environment set ***Initial***, and the output is *Tree* after inserting attribute f. The function of the algorithm is to insert attribute f into *Tree* in a suitable position. First, the algorithm determines whether the parameter value of the HostID type of the attribute f is equal to the identification of a second-tier node in *Tree*. If no such node exists, then the node i with the HostID type parameter value identification of the attribute f is generated as the newly added second-tier node. Then, the node j with the predicate name identifier of the attribute f is generated. The node j acts as the postorder node of the node i, and then the attribute f acts as the postorder node of the node j. If the parameter value of the HostID type of the attribute f equals the identity of the second-tier node k existing already in *Tree*, then it is judged whether the predicate name of the attribute f equals the identity of a postorder node of the node k. If such a postorder node does not exist, then a node m with the predicate name identifier of the attribute f is generated as the postorder node of the node k. If such a postorder node n exists, then the attribute f is used as the postorder node of the node n.

Procedure *CreateAtomicAttacks*(***DM***, ***Initial***)
Input: ***DM***, ***Initial***.
Output: ***AA***.
1. $Tree = \Phi$;
2. $Queue \longleftarrow$ ***Initial***;
3. **For** *each f* **in** ***Initial*** **Do**
4. $T = Pre\text{-}Treatment(Tree, f)$;
5. **While**$(Queue \neq \Phi)$
6. $f = Queue.Dequeue()$;
7. **For** *each DM* **in** ***DM*** **Do**
8. $Ack_f = InstantiatePattern(\boldsymbol{T}, \boldsymbol{AA}, f)$;
9. **For** *each ack* **in** $\boldsymbol{Ack_f}$ **Do**
10. **If** *ack* **in** ***AA*** **Then**.
11. $\boldsymbol{AA} = \boldsymbol{AA} \cup \{ack\}$;
12. **For** *each e* **in** *DM.Eff* **Do**
13. **If** *e* **in** ***T*** **Then**
14. $Queue.Dequeue(e)$;
15. $T = Pre\text{-}Treatment(T, e)$;
16. **For** *each ack* **in** ***AA*** **Do**
17. **For** *each p* **in** *ack.Pre* **Do**
18. $E = E \cup \{Node(p) \longrightarrow Node(ack)\}$;
19. **For** *each e* **in** *ack.Eff* **Do**
20. $E = E \cup \{Node(ack) \longrightarrow Node(e)\}$;
21. **Return** ***AA***.

Algorithm 2: Atomic attack set construction algorithms.

3. Construction of the Invocation Chain and Its Effectiveness Evaluation

The attack graph demonstrates well how to use multiple vulnerabilities in the fog computing system for the combined application of detection patterns [15, 16]. This paper adopts the method of invocation chain, where the program instrumentation is performed in the execution process of each step detection pattern, and events capture and log output are carried out in each path of the vulnerability combination operation, so that the detection effect of each step can be tracked and observed conveniently. In addition, this paper also evaluates the effectiveness of the invocation chain to verify whether the invocation chain technology can detect existing or unknown vulnerabilities.

Procedure *CreateChain*(***AA***)
Input: ***AA***
Output: ***V***
1. $\boldsymbol{V} = \Phi$, $\boldsymbol{V}_{\mathbf{new}} = \Phi$;
2. *Random* = 0;
3. **For** *each* **in** ***AA*** **Then**
4. $V_{\text{new}}.Node = Random + +$;
5. $V_{\text{new}}.S\_id\text{-}D\_id = ack.Pre.netConn.s + \text{“-”} + ack.Pre.netConn.d$;
6. $V_{\text{new}}.cveid = ack.Pre.vulExists.cveid$;
7. $\boldsymbol{V} = \boldsymbol{V} \cup \{\boldsymbol{V}_{\mathbf{new}}\}$;
8. $dot(\boldsymbol{V})$;
9. Return V

Algorithm 3: Invocation chain construction algorithms.

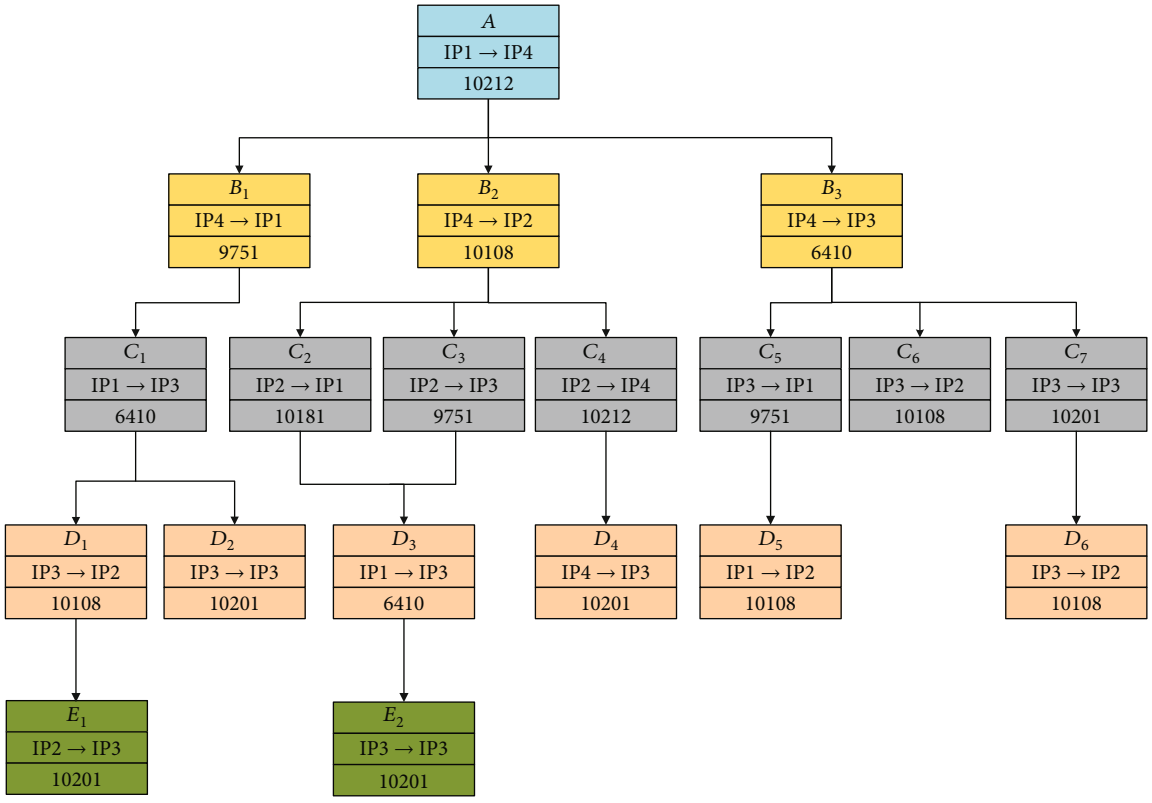

Figure 4: Construction of invocation chain based on the attack graph.

3.1. Construction and Presentation of Invocation Chain. During the execution of the detection pattern, matching checks may be made on the preconditions of each execution. In order to reduce the number of matching checks of preconditions, it is necessary to instantiate the preconditions of detection patterns. Aiming at the system environment of tree structure, the set of atomic attacks $\boldsymbol{Ack}_f$ which can satisfy the preconditions of *DM* is generated by attributes f. An atomic attack is a step-by-step, independent, targeted, and unrepeatable attack, which is taken in a behaviour to achieve the purpose of attack. The attack graph construction algorithm is to generate all executable atomic attack sets for the fog computing system and then convert each atomic attack into *V*(*Node*, *S_id-D_id*, *cveid*) according to *vulExists* (*d*, *sn*, *cveid*) and *netConn* (*s*: *HostID*, *d*: *HostID*, *pro*: *Protocol*, *port*: *Port*), where *Node* is the node number, *S_id-D_id* is the source node and the destination node, and *cveid* is the vulnerability number.

During the execution of the detection pattern, matching checks may be made on the preconditions of each execution. In order to reduce the number of matching checks of preconditions, it is necessary to instantiate the preconditions of detection patterns [17]. For the system environment of tree structure, the set of atomic attacks $\boldsymbol{Ack}_f$ which can satisfy the preconditions of *DM* is generated by attributes f. An atomic attack is an independent, target-oriented, and indivisible one-time attack, which is taken in a behaviour to achieve the purpose of attack. The attack graph

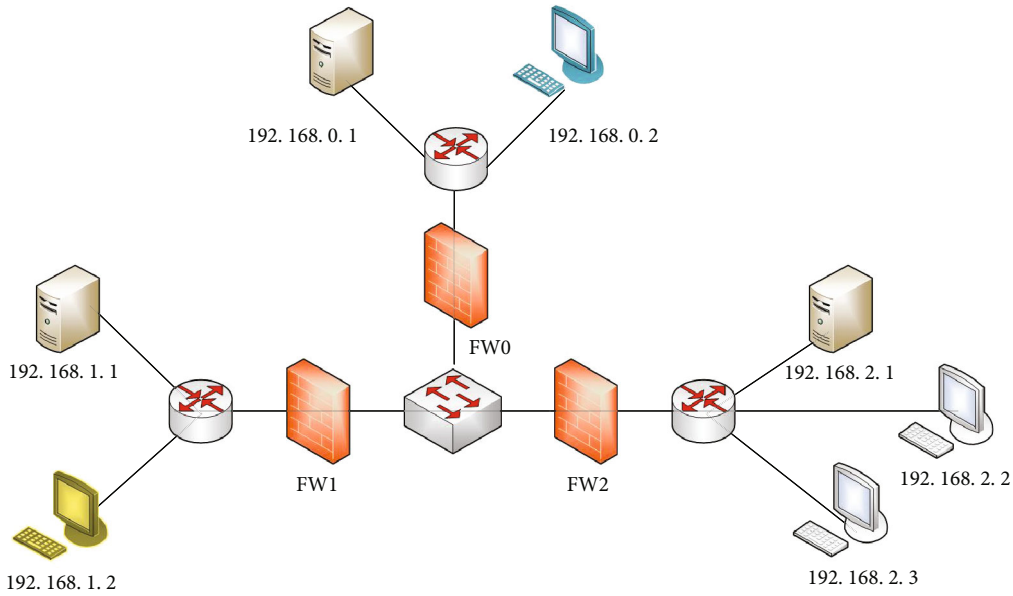

Figure 5: Experimental environment of fog computing.

Table 2: Distribution of vulnerabilities in networks.

Host number	Application	Vulnerability number	Port number
192.168.0.1	tomcat 7.0.39	CVE-2016-4444	80
192.168.0.2	telnet	CVE-2018-0014	23
192.168.1.1	IIS FTP7.5	CVE-2015-3972	21
192.168.1.2	—	—	—
192.168.2.1	PHP, tomcat	CVE-2017-9427	80
192.168.2.2	RPC	CVE-2016-3175	135
192.168.2.3	core ftp	CVE-2017-4643	21

construction algorithm is to generate all executable atomic attack sets for the fog computing system and then convert each atomic attack into *V*(*Node*, *S_id-D_id*, *cveid*) according to *vulExists* (*d*, *sn*, *cveid*) and *netConn* (*s*: *HostID*, *d*: *HostID*, *pro*: *Protocol*, *port*: *Port*), where *Node* is the node number, *S_id-D_id* is the source node and the destination node, and *cveid* is the vulnerability number.

Algorithm 2 is the description of the algorithm for constructing the set of atomic attacks. The input of the algorithm is the correlation detection pattern set ***DM*** and fog computing environment set ***Initial***, and the output is the executable atomic attack set ***AA***. Firstly, the attributes in ***Initial*** are stored in a tree structure ***T***, and a dispatch queue of attributes is established. *Queue.Dequeue*() indicates that the first function in the queue *Queue* is taken out. When the algorithm runs, an attribute f is constantly extracted from the *Queue*, and then the function *InstantiatePattern* is used to generate the set of atomic attacks $\boldsymbol{Ack}_f$ on the attribute f. The consequences of these atomic attacks are added to the tree structure, and the above operations are repeated until the *Queue* is empty. Algorithm 3 is the description of constructing the invocation chain algorithm. The input of the algorithm is ***AA***, which draws the attack graph automatically by calling function dot in Graphviz software [18], and the output of the algorithm is ***V***.

As shown in Figure 4, the constructed invocation chains are presented in the form of attack graphs, and different invocation chains are traversed according to the principle of depth-first [19]. Nodes represent every step in the execution process of invocation chain. Each node is described with three layers. The first layer is composed of letters and numbers which are used to uniquely identify a node vulnerability. The letters represent the top-down level of the node in the attack graph, while the numbers represent the sequence number of the node from left to right in a layer of the attack graph; the second layer represents the source host and target host of the node vulnerability when it is exploited; the third layer indicates the vulnerability number of the node.

3.2. Effectiveness Evaluation of Invocation Chain. In Figure 4, it is assumed that a node is exploited and that there are two or more sequential nodes in the node. The probability of exploiting one of the sequential nodes is P, $P = E * S$. P depends on the probability of availability of the vulnerability itself E and the possibility of this point being selected S. S_{AB} refers to, when there are two or more postorder nodes for node A in the attack graph, the probability of the posternode B being selected under the condition that postorder node A is successfully detected. The possibility of node being selected depends on the availability of vulnerabilities and the distribution of sequential vulnerabilities. The calculation method is shown as

$$S_{AB} = \frac{E_B}{\sum_{k=1}^{N} E_K}. \quad (1)$$

S_i refers to the probability of vulnerability i being selected throughout the attack graph. The calculation method is related to the distribution of the preorder nodes in the attack

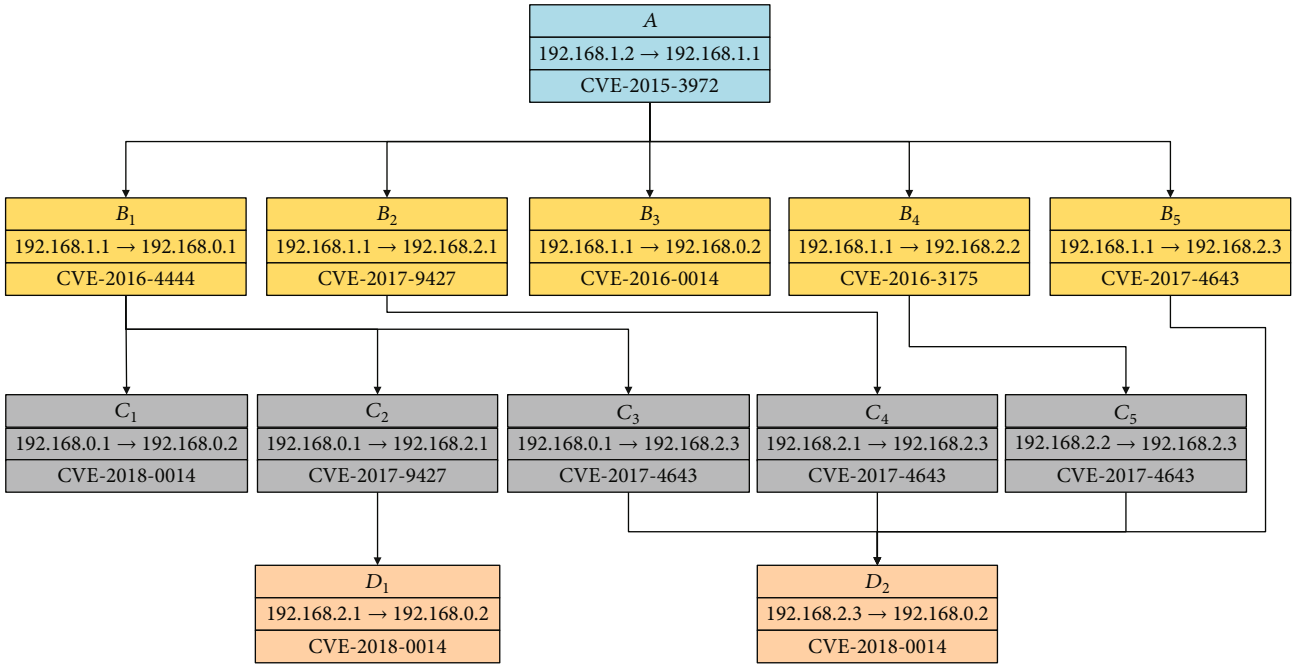

Figure 6: The test instance of invocation chain based on the attack graph.

Table 3: The probability of vulnerabilities being successfully exploited in each path.

Path number	Path description	P_{AVG} (%)
1	$A \longrightarrow B_1 \longrightarrow C_1$	38.60
2	$A \longrightarrow B_1 \longrightarrow C_2 \longrightarrow D_1$	53.63
3	$A \longrightarrow B_1 \longrightarrow C_3 \rightarrow D_2$	67.66
4	$A \longrightarrow B_2 \longrightarrow C_4 \longrightarrow D_2$	64.59
5	$A \longrightarrow B_3$	13.81
6	$A \longrightarrow B_4 \longrightarrow C_5 \longrightarrow D_2$	72.14
7	$A \longrightarrow B_5 \longrightarrow D_2$	49.76

graph. In the attack graph, the distribution of preordered and postordered nodes can be roughly divided into the following two types:

(1) P_A denotes the probability of vulnerability A being successfully exploited. S_B denotes the selectivity of vulnerabilities B after vulnerability A is exploited. It goes like $S_B = P_A S_{AB}$, and the probability of vulnerability B being exploited P_B is $P_B = E_B S_B$

(2) The relationship between the invocation chains from A to C and that from B to C is logic, and P_A presents the probability of vulnerability A being successfully exploited and P_B the probability of vulnerability B being successfully exploited. According to the probability formula, the probability of vulnerability C being successfully exploited is obtained as $P_C = E_C S_C$. The selectivity of vulnerabilities S_C is

$$S_C = P_A S_{AC} * P_B S_{BC}. \tag{2}$$

The probability of a vulnerable node being successfully exploited in a specific invocation chain can be obtained by the above calculating process. However, it is not equal to the probability that the vulnerability will be successfully exploited in the whole network, as the vulnerability may appear in different invocation chains. The comprehensive probability of the vulnerability being successfully exploited is determined by the probabilities of the vulnerability node being successfully exploited in different paths. With the weighted average method [20], the comprehensive probability of vulnerabilities being successfully exploited is calculated as

$$P_{AVG} = \sqrt{\sum_{i=1}^{n} \frac{p_i}{\sum_{i=1}^{n} p_i} p_i} = \sqrt{\frac{\sum_{i=1}^{n} p_i^2}{\sum_{i=1}^{n} p_i}}. \tag{3}$$

In (3), p_i denotes the probability of the vulnerabilities being successfully exploited in their respective test sequences, and n denotes the number of vulnerabilities arising in the whole network attack graph. According to the different P_{AVG}, the degree of vulnerability threat is divided into five levels, i.e., security [0, 20%), low threat [20%, 40%), medium threat [40%, 60%), high threat [60%, 80%), and emergency [80%, 100%] [21].

4. Simulation Results

As shown in Figure 5, the experimental network topology is divided into three segments, i.e., 192.168.0.0/24, 192.168.1.0/24, and 192.168.2.0/24, termed as Segments 0, 1, and 2, respectively. This model can overcome the shortcomings of cloud computing in location awareness, big data exploitation, real-time content delivery, low service delay,

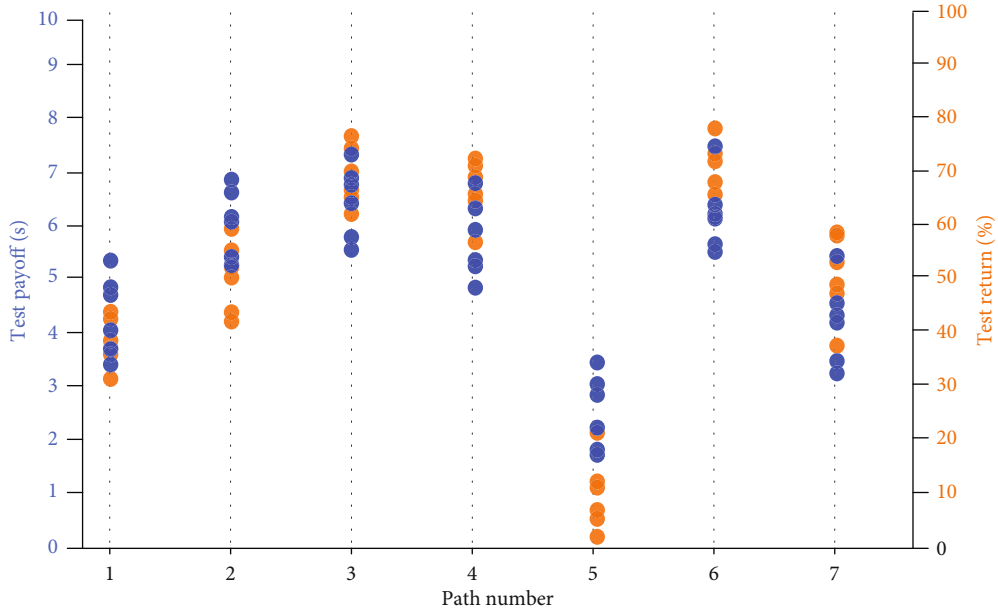

Figure 7: The comparison of test payoff and test effort.

and so on [22]. The network segments are isolated by firewalls. Port filtering and IP filtering rules are configured on firewalls and routers, respectively, to control the accessibility between network segments. The host 192.168.1.2 is the origin of the invocation chain. The host 192.168.0.2 is the final target host, and the target of attack is to gain its system privileges [23, 24]. The distribution of vulnerabilities information of each host in the fog computing system is shown in Table 2.

The test instance of invocation chain method based on the attack graph is shown in Figure 6. Due to the RPC vulnerability of 192.168.2.2 in segment 2, the path from 192.168.2.3 to 192.168.2.2 will be added to the attack graph. Because CVE-2017-4643 vulnerability exists in core FTP software of 192.168.2.3 in network segment 2, the testers on remote FTP server can execute arbitrary code with the help of long strings. The FTP server on 192.168.1.1 server can exploit this vulnerability to attack, and the attack graph will be added to the path from 192.168.1.1 to 192.168.2.3.

According to the effectiveness evaluation of invocation chain mentioned above, the successful probability of atomic attack and P_{AVG} in each path can be calculated, respectively. The results of the calculation are shown in Table 3. The comparison results of invocation chain tests in each path are $P_{\mathrm{AVG}}(\mathrm{Path}\,5) < P_{\mathrm{AVG}}(\mathrm{Path}\,1) < P_{\mathrm{AVG}}(\mathrm{Path}\,7) < P_{\mathrm{AVG}}(\mathrm{Path}\,6) < P_{\mathrm{AVG}}(\mathrm{Path}\,3) < P_{\mathrm{AVG}}(\mathrm{Path}\,2) < P_{\mathrm{AVG}}(\mathrm{Path}\,4)$. Path 4 is the most vulnerable path to detect system vulnerabilities, and it is also the most important to strengthen the protection of this path to prevent network attacks.

Figure 7 shows the comparison of test payoff and test effort. Test payoff refers to the times to scan and analyse network vulnerabilities, and test return refers to the probability of the vulnerabilities being successfully exploited in the fog computing system through invocation chain tests. For security testing, we hope to choose the path with the minimum test payoff and the maximum test effort [25]. When evaluating the system security and deciding the security policy, we should also consider that the attacker may take advantage of this behaviour. Although test payoff of Paths 2, 3, 4, and 6 is high, they can also get high test return. The P_{AVG} values of the above four test sequences are higher than 50%. The shortest test time required for testing according to Path 5 is basically within 5 seconds, and its P_{AVG} value is 13.81%.

As can be seen from Figure 8, due to the different P_{AVG} values of each test sequence, the test payoff and test effort in each path are also different. In order to integrate the impact of payoff and effort on security test decision, the multiobjective decision-making method can be used to analyse which path is more efficient and faster to determine the test sequence under the given test environment [26].

In the above fog computing experimental environment, this paper compares the alarm time of vulnerability detection with that of the model-based vulnerability assessment method proposed in [6] and the attribute-based vulnerability assessment method proposed in [7] and analyses the impact of different methods on the alarm time of vulnerability detection. The faster the alarm time of vulnerability detection, and the earlier the security flaws can be found in the fog computing system [27]. Figure 8 shows the comparison results of various methods under the premise that the experimental environment and parameters are basically the same. It can be seen that the invocation chain method proposed in this paper is superior to other methods in mapping time and association analysis time, because other methods need mapping and exchange from alarm to node, node to path, and path to processing unit; meanwhile, association analysis needs to analyse other alarms on all paths where the alarm is located, and there will be overlaps on paths and repeated alarm message processing. A large amount of repeated analysis in the multinode and multiconnection fog computing

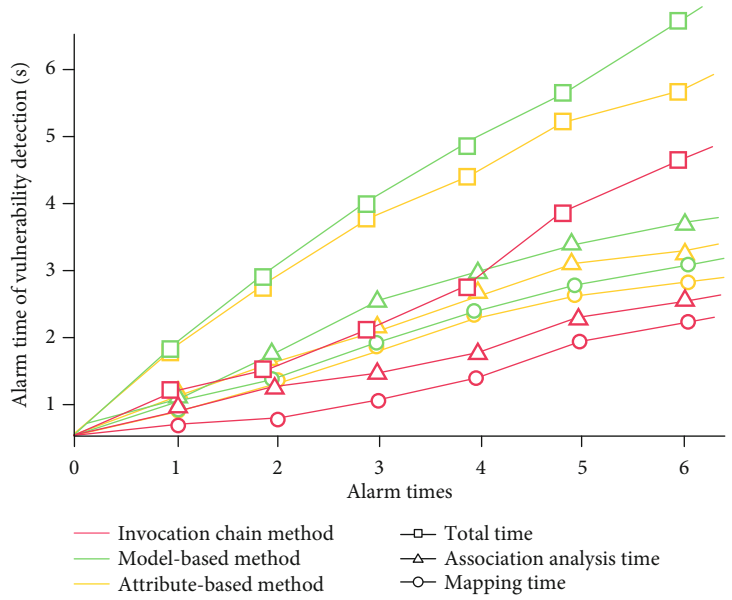

Figure 8: The comparison results of the alarm time of vulnerability detection.

environment obviously increases the cost of alarm processing time. In Figure 8, when alarm time is 2, the mapping time of the invocation chain method is about 47% lower than that of the model-based vulnerability assessment method, and the association analysis time is about 38% lower than that of the attribute-based vulnerability assessment method.

5. Conclusion

Fog computing systems face various security threats, and the vulnerabilities in distributed fog nodes restrict the development of fog computing technology. In the paper, the invocation chain test method proposed can not only analyse the impact of nodes' vulnerabilities but also analyse the impact of test sequence on fog computing security. By analysing the network topology of fog computing and the relationship among the fog nodes exploited successively, the importance of test sequence in the network is obtained. The higher the probability that the node's vulnerability is selected, the greater the impact on network security; the higher the probability that a vulnerable node is being successfully expanded, the more security protection is needed. Through the test results, it is easy to find that the test chain method can better find potential or unknown vulnerabilities, and the probability of vulnerability successfully exploited is generally raised to more than 50%. The method of invocation chain is used to vulnerability assessment. The invocation chain generated based on the attack graph is concise and clear, which can be better applied to complex large-scale systems.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Authors' Contributions

Y. Zhao and Y. Yang contributed equally to this study.

Acknowledgments

This work was supported by the Applied Basic Research Project of Sichuan Province (No. 2018JY0370) and the Key Research and Development Project of Sichuan Province of China (No. 2020YFG0292).

References

[1] G. Zhu, Y. Yin, R. Cai, and K. Li, "Detecting virtualization specific vulnerabilities in cloud computing environment," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 743–748, Honolulu, CA, USA, 2017.

[2] A. Sun, G. Gao, T. Ji, and X. Tu, "One quantifiable security evaluation model for cloud computing platform," in *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, pp. 197–201, Lanzhou, China, 2018.

[3] S. Tu, M. Waqas, S. U. Rehman et al., "Security in fog computing: a novel technique to tackle an impersonation attack," *IEEE Access*, vol. 6, pp. 74993–75001, 2018.

[4] D. Puthal, S. P. Mohanty, S. A. Bhavake, G. Morgan, and R. Ranjan, "Fog computing security challenges and future directions [energy and security]," *IEEE Consumer Electronics Magazine*, vol. 8, no. 3, pp. 92–96, 2019.

[5] C. Bozzato, "Vulnerability spotlight: multiple vulnerabilities in samsung smartthings hub," July 2018, https://blog

.talosintelligence.com/2018/07/samsung-smartthings-vulns.html.

[6] J. N. Goel, M. H. Asghar, V. Kumar, and S. K. Pandey, "Ensemble based approach to increase vulnerability assessment and penetration testing accuracy," in *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*, pp. 330–335, Noida, India, 2016.

[7] C. Ji, P. Yu, W. Li, P. Zhao, and X. Qiu, "Comprehensive vulnerability assessment and optimization method for smart grid communication transmission systems," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 1238–1245, Lisbon, Portugal, 2017.

[8] M. S. Barik, "AGQL: a query language for attack graph-based network vulnerability analysis," in *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, pp. 1–4, Kolkata, India, 2017.

[9] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino, "Kalis-a system for knowledge-driven adaptable intrusion detection for the Internet of Things," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 656–666, Atlanta, GA, USA, 2017.

[10] C. Gong, F. Lin, X. Gong, and Y. Lu, "Intelligent cooperative edge computing in the Internet of Things," *IEEE Internet of Things Journal*, vol. 99, pp. 1–11, 2020, to be published.

[11] H. Hui, C. Zhou, S. Xu, and F. Lin, "A novel secure data transmission scheme in industrial Internet of Things," *China Communications*, vol. 17, no. 1, pp. 73–88, 2020.

[12] I. Kotenko and E. Doynikova, "The CAPEC based generator of attack scenarios for network security evaluation," in *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, pp. 436–441, Warsaw, Poland, 2015.

[13] B. Vandeportaele, "A finite state machine modeling language and the associated tools allowing fast prototyping for FPGA devices," in *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, pp. 1–6, Donostia-San Sebastian, Spain, 2017.

[14] H. S. Lallie, K. Debattista, and J. Bal, "An empirical evaluation of the effectiveness of attack graphs and fault trees in cyber-attack perception," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1110–1122, 2018.

[15] J. Yang, T. Li, G. Liang, W. He, and Y. Zhao, "A simple recurrent unit model based intrusion detection system with DCGAN," *IEEE Access*, vol. 7, no. 1, pp. 83286–83296, 2019.

[16] J. Yang, T. Li, G. Liang, W. He, and Y. Zhao, "A hierarchy distributed-agents model for network risk evaluation based on deep learning,," *Computer Modeling in Engineering & Sciences*, vol. 120, no. 1, pp. 1–23, 2019.

[17] Y. Zhao, X. Fang, R. Huang, and Y. Fang, "Joint interference coordination and load balancing for OFDMA Multihop cellular networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 1, pp. 89–101, 2014.

[18] Y. Shang, "Finite-time weighted average consensus and generalized consensus over a subset," *IEEE Access*, vol. 4, no. 8, pp. 2615–2620, 2016.

[19] J. Yi, T. Clausen, and U. Herberg, "Depth-first forwarding for unreliable networks: extensions and applications," *IEEE Internet of Things Journal*, vol. 2, no. 3, pp. 199–209, 2015.

[20] K. Huang, C. Zhou, Y. Qin, and W. Tu, "A game-theoretic approach to cross-layer security decision-making in industrial cyber-physical systems," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 3, pp. 2371–2379, 2020.

[21] S. Wang, G. Tang, G. Kou, and Y. Chao, "An attack graph generation method based on heuristic searching strategy," in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 1180–1185, Chengdu, China, 2017.

[22] Y. Wang, L. Xie, W. Li, W. Meng, and J. Li, "A privacy-preserving framework for collaborative intrusion detection networks through fog computing," in *International Symposium on Cyberspace Safety and Security*, pp. 267–279, Springer, 2017.

[23] Y. Zhao, Z. Chen, H. Su, E. Sun, Y. Guo, and J. Ding, "A position-based secure fast handover mechanism for high-speed trains," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pp. 1–6, Guangzhou, China, 2018.

[24] M. S. Khazaei, H. Homaei, and H. R. Shahriari, "OPEXA: analyser assistant for detecting over-privileged extensions," *IET Information Security*, vol. 12, no. 6, pp. 558–565, 2018.

[25] M. Barni and B. Tondi, "Binary hypothesis testing game with training data," *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 4848–4866, 2014.

[26] Z. Shan, H. Wang, G. Ying, B. Zhang, and J. Xu, "Multi-objective decision-making model for distribution system condition-based maintenance," in *2016 China International Conference on Electricity Distribution (CICED)*, pp. 1–5, Xi'an, China, 2016.

[27] H. AlTair, T. Taha, J. Dias, and M. Al-Qutayri, "Decision making for multi-objective multi-agent search and rescue missions," in *2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, pp. 1–4, Ras Al Khaimah, United Arab Emirates, 2017.