TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 10/13 pp508-515
Volume 19, Number 5, October 2014

Malicious Code Detection Model Based on Behavior Association

Lansheng Han, Mengxiao Qian*, Xingbo Xu, Cai Fu, and Hamza Kwisaba

Abstract: Malicious applications can be introduced to attack users and services so as to gain financial rewards, individuals' sensitive information, company and government intellectual property, and to gain remote control of systems. However, traditional methods of malicious code detection, such as signature detection, behavior detection, virtual machine detection, and heuristic detection, have various weaknesses which make them unreliable. This paper presents the existing technologies of malicious code detection and a malicious code detection model is proposed based on behavior association. The behavior points of malicious code are first extracted through API monitoring technology and integrated into the behavior; then a relation between behaviors is established according to data dependence. Next, a behavior association model is built up and a discrimination method is put forth using pushdown automation. Finally, the exact malicious code is taken as a sample to carry out an experiment on the behavior's capture, association, and discrimination, thus proving that the theoretical model is viable.

Key words: malicious code; behavior monitor; behavior association; pushdown automation

1 Introduction

With the rapid development of computer technology, the computer has penetrated every aspect of people's lives but, while the openness and flexibility of computers bring convenience, with this come various security risks. Malware such as viruses, trojans, and botnets has caused great financial loss to many individuals, companies, and governments and becomes one of the major factors that threaten internet security. As the security threat becomes more and more serious, a deeper understanding of trojans[1], viruses[2], and botnets[3] has been required and some advances have been made.

Popular malicious code detection methods including signature detection, behavior detection, virtual machine detection, heuristic detection, etc., have their drawbacks. Signature detection can only identify known malicious code; it does not work on unknown malicious code[4,5]. Behavior detection depends too much on program execution so its discrimination may be wrong if the execution conditions are not met[6,7]. Virtual machine detection can generally detect encrypted malicious code but it needs the help of signature scanning and, furthermore, code with special instructions may escape virtual machine detection[8]. Heuristic detection can discriminate unknown malicious code, but it has complicated judgment logic, difficult development, and a high false positive ratio[9,10].

Analyzing the various malicious code detection technologies and assessing their respective pros and cons, we find that all the above detection technologies have one thing in common: they are all focused on the local characteristics of malicious code. Needless to say, local characteristics are not unique to malicious code. In addition as hackers improve their malware writing skills, the characteristics of malicious code gradually change, making antivirus software which focuses on local characteristics have a higher false positive ratio. This has therefore made it necessary to

• Lansheng Han, Mengxiao Qian, Xingbo Xu, Cai Fu, and Hamza Kwisaba are with the Lab. of Information Security, School of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, China. Email: hanlansheng@hust.edu.cn; 2570759136@qq.com; xuxingbo@hust.edu.cn; stand_fucai@126.com; kwiha@mail.ru.

∗ To whom correspondence should be addressed.

Manuscript received: 2014-04-15; revised: 2014-07-15; accepted: 2014-08-25

detect malicious code's whole behavior.

Accordingly, we put forward a detection method based on behavior association. In this method, we focus on the abstract description of malicious behavior with the intent to clarify the relationship between behaviors rather than considering the maliciousness of a single behavior. The purpose is to analyze the malicious code comprehensively and to increase the rate of discrimination of malicious code.

In summary we make the following contributions:

• For the first time we propose a new method for malware detection using behavior association.

• We provide a practical implementation of our newly proposed model and through the experiments performed, it proves to be reliable.

• We also study and show the weaknesses of currently used malware detection technologies.

2 Behavior Association Model

2.1 The definition of behavior association

2.1.1 Behavior point

Definition 1 A behavior point is an API that the program calls in order to finish a function.

From classifying Windows API, we get 5 categories: *file behavior point*, *registry behavior point*, *process behavior point*, *web behavior point, and other behavior point*.

2.1.2 Behavior

Definition 2 Behavior is a set of one or multiple relatively close behavior points, such as reading file behavior {OpenFile, ReadFile} or ending process behavior {OpenProcess, TerminateProcess}. To facilitate the description of the subsequent algorithm, the formal description is as follows:

The behavior is described by 5-tuples $A=(S,I,O,T,C)$, where S represents the behavior states (execution results), such as success, failure or other states; I represents the input of the behavior; O represents the output of the behavior; T represents the type of behavior, such as file behavior, process behavior; C represents the set of behavior points that combine the behavior.

2.1.3 Behavior association

Definition 3 Behavior association is the relationship between behaviors, mainly referring to dependence relationships.

For example, behavior A represents reading file, behavior B represents writing file, the content written in file(B) is from reading file(A); specifically, the input of behavior B comes from the output of behavior A($A.O \rightarrow B.I$), which means that behavior B depends on behavior A, written as $A \rightarrow B$.

2.2 The digraph of behavior association

Since the relationship between behaviors is a dependence relationship, digraph $G(V,E)$ can be used to represent behavior association. A node V_i represents a behavior, a directed edge (V_i,V_j) represents the dependence relationship, while $V_i \rightarrow V_j$ represents V_j which depends on V_i.

Here, we take an example trojan to establish a behavior association digraph.

The trojan's main functions are as follows: (1) open some ports of the computer; (2) communicate with the remote host; (3) steal important information (username, password, etc.) from the target computer; (4) monitor the desktop of the target computer; (5) run on start up.

The trojan's execution process is as follows: add starting up work item$\Longrightarrow$open a port$\Longrightarrow$communicate with remote host$\Longrightarrow$monitor the desktop of target computer$\Longrightarrow$steal important information from target computer$\Longrightarrow$start a new process$\Longrightarrow$end process.

First, we extract the following behaviors from the behavior points: ① work after starting up; ② open a port; ③ receive connection from remote host; ④ receive information from remote host; ⑤ send information to remote host; ⑥ start process; ⑦ end process; ⑧ read file; ⑨ screen-shot.

Then, we establish behavior association. It is obvious that ①, ②, ⑧, and ⑨ do not depend on other behaviors, and that the remaining behaviors' dependence relationships are as follows: {②→③, ③→④, ③→⑤, ④→⑤, ⑤→④, ⑧→⑤, ⑨→⑤, ④→⑥, ④→⑦}. The behavior association digraph is shown in Fig. 1.

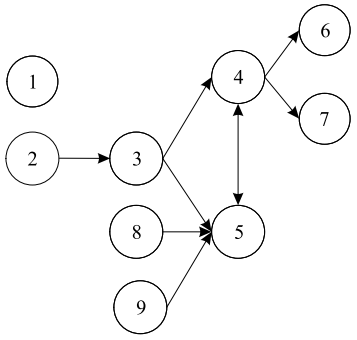

Fig. 1 Behavior association digraph.

From Fig. 1, we can clearly see the trojan's behavior association so its identification is not difficult, but establishing the behavior associations is the key problem that we must solve.

2.3 Behavior association establishment

2.3.1 Behavior extraction

Behavior extraction is the integration of monitored behavior points into a behavior.

To do this, we first put the behavior points into the corresponding category, then deal with integration within the category. Behavior point integration relies on the kernel object[11]. When a function that creates the kernel object is called, a handle representing the object is returned by the function so, if we want to manipulate the kernel object, we need to send the handle to the API function so that the behavior can be extracted through the handle. Handles can be divided into five categories according to the behavior point's classification; for each category, a linked list is used to save handles. When a new handle is created, it is taken to the corresponding linked list. If we need to analyze the behavior point that manipulates the handle, we request it from the corresponding linked list, then integrate the behavior points which have called the same handle into a behavior, and thus the process of behavior extraction is finished.

2.3.2 The establishment of relationship between behaviors

After behavior extraction, we then establish the relationship between behaviors.

The relationship between behaviors includes data and non-data dependence; in this paper we focus on the former. Data dependence means the ordinal relations in data processing, such as transitive relations. For example, when document B reads document A's content, it is a typical data transitive relation. This relationship is relatively close so the behavior association could be established by data dependence. Non-data dependence mainly refers to the dependence of logical structure: for example, behavior A and behavior B access or modify different fields in the same database file. In comparison to data dependence, non-data dependence has a looser relationship and behavior association can not therefore be established.

From the above discussion, it is clear that the relationship between behaviors can be obtained by tracking the programs that the data file accesses. For example, we can record the behavior's data address (internal memory or buffer) and its change, and observe whether the data address is called by other behaviors. With this, we can determine the dependence between behaviors from the data's transmission and establish a behavior association digraph based on Section 2.2.

The record of behavior buffer is shown in Fig. 2 and detailed algorithm is as follows:

For each collected behavior A_i do:
  Record related buffer $B_{i,k}$ from $k = 1$ to $k = m$ under A_i
For new collected behavior A_j
  For all collected behavior A_i do:
    For all related buffer $B_{i,k}$
      From $k = 1$ to $k = m$ do:
        Compare ($B_{j,1}$, $B_{i,k}$)
    Next
  Next
If $B_{i,k}$ equals $B_{j,1}$
  Then $A_i \rightarrow A_j$
Else
  Record related buffer $B_{j,k}$ under A_j

The relationship between behaviors is mostly transmitted by parameters while a minority is transmitted by commands without parameters, such as the shutdown command. In most cases, behavior B can call a related parameter of behavior A, thus manipulating the data which behavior A manipulated, further establishing the data relationship between the two behaviors. In other words, the analysis of parameters can help us understand more about the relationship between behaviors. Take the address parameter for example: A buffer is a kind of typical address parameter. When we analyze the buffer address, we can regard address a_v and the address space s_v to which α_v points as the same parameter, so that address a ($a_v \leqslant a \leqslant s_v$) is a parameter. To analyze the address parameter better, we can classify some addresses with important attributes such as import table, service description table, entry point, etc. A decision tree can be established based on address space, as shown in

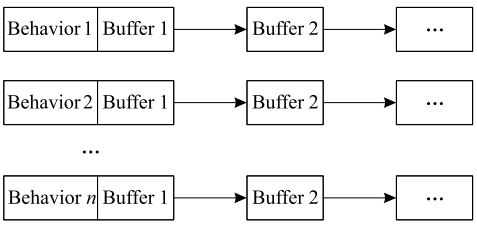

Fig. 2 Record of behavior buffer.

Fig. 3, so we can analyze different address spaces better with the help of a decision tree[12].

3 The Grammar Description and Discrimination of Behavior Association

Once the behavior association digraph is established, we can describe the behavior association using the behavior's grammar, so that detecting malicious behavior can become a question of analyzing this grammar which can be done by automation.

3.1 The grammar description of malicious behavior

We take a typical trojan as an example to show the malicious behavior's grammar description. According to the analysis of the trojan's main behaviors in Section 2.2 and with the help of grammatical analysis[13], we give the following grammar description.

3.1.1 Grammar of duplication behavior

Duplication behavior can be divided into three types: mono-block read/write, intersecting read/write, and direct-copy.

(1)<Duplicate>::=<Create><Open><Read><Write>
// "::=" means "define"
|<Open><Create><Read><Write>
// "|" means "or"
|<Open><Read><Create><Write>
// "<>" represents a grammatical unit
|<Open><Create><IntrlvRW>
|<Create><Open><IntrlvRW>
|<DirectCopy>;
(2) <Read>::=receive object1←object2;
(3) <Write>::=send object1→object2;
(4) <IntrlvRW>::=while(receive object1←object2){
send object3→object4;}

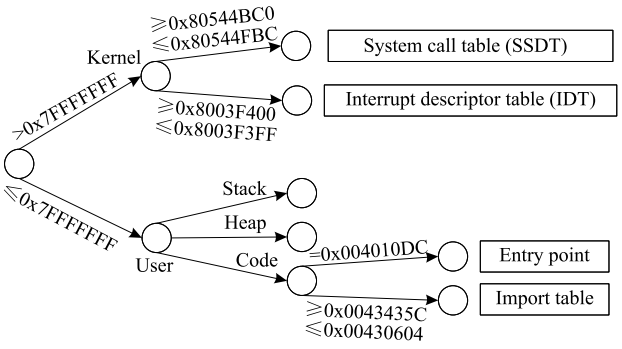

Fig. 3 Address analysis.

(5)<DirectCopy>::=send object1→object2.

3.1.2 Transitive behavior grammar

(1) <Propagate>::=<Open><Read><Transmit>
|<Read><Open><Transmit>
(2)<Transmit>::=<Format><Write>|<Write>.

3.1.3 Infection behavior grammar

(1) <Infection>::=<Search><Open><Relocate><Read>
<Mutation><Write>
|<Search><Open><Read><Relocate>
<Mutation><Write>
(2) <Search> ::= while(find object){
open object1 ;
receive object2←object1;}
(3) <Relocate> ::= receive object2←object1;
send object2→object1;
| null.

3.2 Pushdown automation

After finishing the grammar description of behavior association, we can establish a state transition diagram with states, behavior, and grammar description as input. This can be done by automation.

In comparison to finite automation, pushdown automation has a stack architecture which can be used to record the function call and the data returned. This makes it more precise than finite automation and is why we choose pushdown automation to establish the model.

In PushDown Automation (PDA)[14, 15] M can be describe by a 7-tuples:

$$M=(Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ Z_0,\ F),$$

where Q is a finite set of states. $\forall q \in Q$ is a state in Q, means behavior association states; Σ is the input alphabet, means a set of malicious behaviors and behavior grammar; Γ is the stack alphabet, $\forall T \in \Gamma$, T is a stack symbol; $Z_0 \in \Gamma$ is a start symbol; $q_0 \in Q$ is the initial state of M, means the initial behavior state; $F \in Q$ is a set of final states, means the final state the pushdown automation can reach, which means the behavior detected is malicious; δ is a transition function.

$$\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \to 2^{Q\times T^*};$$
$$\forall (q, a, Z) \in Q \times \Sigma \times \Gamma,$$

$\delta(q, a, Z) = \{(p_1, r_1), (p_2, r_2), \cdots, (p_m, r_m)\}$ means that the PDA is in state q, with input symbol a and the top symbol Z on the stack alphabet. For any i, entering state p_i, replaces symbol Z by string r_i from right to left, and advances the input head one symbol.

3.3 The process of discrimination

We can configure a series of pushdown automation based on numerous samples of malicious code. If there are behaviors which need to be detected, we can use pushdown automation to discriminate them with the behaviors as input. If the final state can be reached by pushdown automation it means the behavior is malicious. The automation model is as shown in Fig. 4.

The detailed algorithm is as follows:

Behavior Detection $(A_1, \cdots, A_n)$
where A_i are collected behaviors (input symbol combined with semantic values: $\{\Sigma \cup \varepsilon\}$)
  For each collected behavior A_i do:
    For all the automation M_k
    from $k = 1$ to $k \leqslant n$ do:
    For all state and stack $(Q_{k,j}, \Gamma_{k,j})$
      from $j = 1$ to $j \leqslant m$ do:
      M_k.ll-parse $(A_i, (Q_{k,j}, \Gamma_{k,j}))$
      Next
    Next
  Next
M.ll-pares$(A, (Q, \Gamma))$ If A, Q, Γ match a transition $T \in \delta_M$ Then
    If Q is an accepting state $Q \in F_M$
      Then malicious behavior detected
    Else
      Change the current state
  Else
    Ignore A

---

4 Experiment

The subject experiment is done on a virtual machine running Windows XP SP3 with Microsoft Visual C++ 6.0 as the development tool.

The experiment detects the same virus sample using both popular commercial antivirus solutions and the behavior association model proposed in the paper. We then compare and analyze the results from which we find the model to be viable. The sample used in the lab for the experiment contains a keyboard hook module, a virus self-replicating module, a repeated infection prevention module, and a self-hiding module.

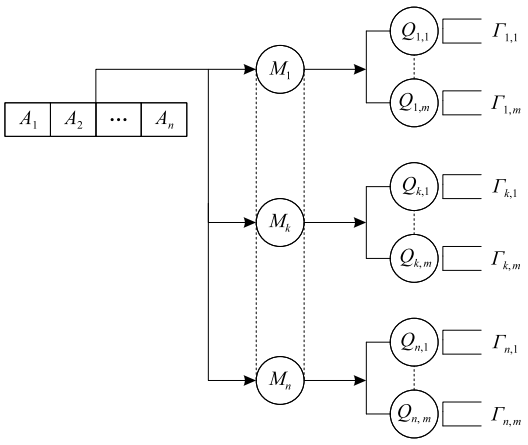

Fig. 4 The model of pushdown automation.

Experiment procedure is as follows:

(1) Monitor API by API monitoring module;

(2) Extract behavior according to API monitored;

(3) Establish behavior association digraph, as shown in Table 1.

Table 1 shows us the capture of behavior and the establishment of relationship. It can be seen that behavior points of malicious code are not obviously different from those of a normal application, but the malicious element gradually manifests itself. The key technology of integrating behavior points into behavior is in the fourth column of the table that is obtaining the association of behavior points by parameters. If the dependence relationship cannot be obtained, the subsequent discrimination will be difficult.

(4) Establish pushdown automation according to the virus' behavior association, as shown in Table 2.

Table 2 shows the discrimination of associated behavior by pushdown automation. From the table, we can see that it is possible to carry out malicious discrimination by pushdown automation, but the premise is that we collect numerous and typical samples of malicious code.

(5) Discriminate behavior using pushdown automation and other antivirus software as shown in Table 3.

To determine the viability of the behavior association discrimination technique, we choose 15 malicious codes, four of which have been very well known during the last few years: ZeuS (record bank details of customers by keystroke logging and form grabbing), Reveton (extort money by locking user's computer), Red October (attack computers of diplomatic institutions) and Trojan.Generic (steal the password). The remaining 11 malicious codes are undisclosed among which the first eight are trojans whose aim is to steal the target's information, such as collecting the IP address, hooking the keyboard, etc. The last three are worms which replicate themselves in order to spread to other computers, as well as collect the target's routing information.

From Table 3 we can see that the popular commercial antivirus solutions can only give inconclusive warnings about the undisclosed malicious code. On the contrary,

Table 1 Behavior and dependence relationship.

Behavior point	Behaviors extraction	Behavior association	Related parameter explanation
GetCurrentDirectoryA	V1: Get the directory	V1→V2	Transmitted by buffer 0x0013FE30
CreateFileA	V2: Open the target file	V2→V3	Transmitted by file handle 00000010
		V2→V4	Transmitted by file handle 00000010
		V2→V6	Transmitted by file handle 00000010
CreateFileMappingA MapViewOfFile	V3: Map file		
SetFilePointer	V4: Require file pointer	V4→V5	Transmitted by file buffer
WriteFile	V5: Write file	V5→V6	Transmitted by file handle 00000010
SetEndOfFile FlushViewOfFile UnmapViewOfFile CloseHandle	V6: Close file		
GetCurrentDirectoryA	V7: Require system directory	V7→V8	Transmitted by directory buffer 0013FC28
CopyFileA	V8: Copy file to system directory		
CreateProcessA	V9: Create process	V9→V10	Transmitted by process handle
GetModuleFileNameA CloseHandle	V10: Program self-detects		
ExitProcess	V11: Quit program	V11→V10	Transmitted by process handle

Table 2 Description of pushdown automation.

State alphabet	Input alphabet	Stack alphabet	Transitive function	Explanation
Q0: starting program state	V1,V2	Z0	δ(Q0,{V1,V2},Z1)={(Q1,Z1)}	Reaching opening file state by opening file
Q1:opening file state	V4,V5,V6	Z1	δ(Q1,{V4,V5},Z1)={(Q2,Z2)}	Reaching writing file state by writing data into file
			δ(Q1,{V6},Z1)={(Q3,Z5)}	Reaching closing file state by closing file directly after opening the file
Q2:writing file state	V5,V6	Z2, Z3	δ(Q2,{V5},Z2)={(Q2,Z3)}	Staying in writing file states by inputting writing file behavior
Q3:closing file state	V7,V8	Z4, Z5	δ(Q3,{V7,V8},Z4)={(Q4,Z6)}	Copying by duplication grammar
Q4:copying file state	V9	Z6	δ(Q4,{V9},Z6)={(Q5,Z7)}	Reaching creating process state by creating process
Q5:creating process state	V10,V11	Z7, Z8	δ(Q5,{V10},Z7)={(Q5,Z8)}	Staying in creating state if the program self-detects
			δ(Q5,{V11},Z7)={(Q6,Z9)}	Reaching the final state by quitting program
Q6:ending program state		Z9		

however, the method we propose can conclusively identify malicious code and this proves the method's viability. The disadvantage of our method is that the capture and analysis take too much time. However, we believe that the time cost will decrease with the development of theory and technology.

5 Conclusions

Looking at the current detection of malicious code technologies being used, we point out that the problem of discrimination of malicious code, which is the premise of identification, is not solved. Malicious code today is different from that of the past, which had obvious characteristics. Current antivirus solutions work by downloading suspicious behavior from everyday computer users from which they extract the signatures which are used to catch the malware. Malicious code nowadays has a stronger sense of purpose and pertinence; it therefore will not randomly infect an everyday users' computer, which makes the discrimination process difficult.

For this reason, we propose a malicious code detection method based on behavior association. First, the definitions of behavior points and behavior are given, and then malicious behavior is described by 5-tuples, after which classification of the relationship between behaviors is done. The behavior association is shown by a relationship digraph while

Table 3 Comparison of behavior association method with existing antivirus methods.

Malicious code	Qihoo 360	King soft	Kasper sky	Behavior association method
Key.Trojan.a	?	?	?	√
Vidio.Trojan.win	!	?	?	√
Pswd.Trojan.t	?	?	!	√
Getkbd.Trojan.W	?	?	!	√
Cross.Trojan.net	!	!	!	√
Pswget.Trojan.b	!	!	!	√
IPget.Trojan.win	!	√	√	√
Fish.Trojan.net	!	!	!	√
Crem.Worm.b	!	√	!	√
Netloc.Worm.hns	?	?	?	√
Joke.Worm.wh	!	?	!	√
ZeuS	√	√	√	√
Reveton	√	√	√	!
Red October	√	√	√	√
Trojan.Generic	√	√	√	!

Notes: Symbols "?", "!", and "√" denote "unknown", "warning", and "identified", respectively.

a detailed method of catching the relationship is also provided. Next, the malicious behavior association is described by the behavior's grammar and a discriminate model based on pushdown automation is proposed, according to the behavior association and grammar description. Finally, typical malicious code is used to carry out the experiment, with the experimental results demonstrating that the model is a viable solution.

Acknowledgements

This research was supported by the National Natural Science Foundation of China (Nos. 61272033 and 61272405).

References

[1] F. Cohen, Computer viruses: Theory and experiments, *Computers & Security*, vol. 6, no. 1, pp. 22-35, 1987.

[2] D. Spinellis, Reliable identification of bounded-length viruses is NP-complete, *IEEE Transactions on Information Theory*, vol. 49, no. 1, pp. 280-284, 2003.

[3] B. Stone-Gross, M. Cova, B. Gilbert, R. Kemmerer, C. Kruegel, and G. Vigna, Analysis of a botnet takeover, *IEEE Security & Privacy*, vol. 9, no. 1, pp. 64-72, 2011.

[4] M. Feng and R. Gupta, Detecting virus mutations via dynamic matching, in *IEEE International Conference on Software Maintenance*, Edmonton, Alberta, Canada, 2009, pp. 105-114.

[5] J. R. Harrald, S. A. Schmitt, and S. Shrestha, The effect of computer virus occurrence and virusthreat level on antivirus companies' financial performance, in *Engineering Management Conf., 2004. Proceedings. IEEE International*, 2004, vol. 2, pp. 780-784.

[6] Z.-P. Kang, H. Xiang, and L. Fu, Attack and defence on API hook technology of trojan horse, *Information Security and Communications Privacy*, vol. 2, pp.145-148, 2007.

[7] L. Wang, Y. Li, and Z. Li, A novel technique of recognising multi-stage attack behavior, *Int. J. of High Performance Computing and Networking*, vol. 6, no. 3/4, pp. 174-180, 2010.

[8] J. E. Smith and R. Nair, The architecture of virtual machines, *Computer.*, vol. 38, no. 5, pp. 32-38, 2005.

[9] L. C. Briand, J. Feng, and Y. Labiche, Experimenting with genetic algorithms and coupling measures to devise optimal integration test orders, in *Software Engineering with Computational Intelligence*, T. M. Khoshgoftaar, Ed. Kluwer Academic Publishers, 2003, pp. 204-234.

[10] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, Data mining methods for detection of new malicious executables, in *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 2001, pp. 38-49.

[11] F. Porikli and O. Tuzel, Multi-kernel object tracking, in *IEEE International Conference on Multimedia and Expo*, Amsterdam, Holland, 2005, pp. 1234-1237.

[12] M. Boulif and K. Atif, Multiobjective cell formation with routing flexibility: A graph partitioning approach, *Int. Journal of Computational Science and Engineering*, http://www.inderscience.com/info/ingeneral/forthcoming.php?jcode=ijcse, forthcoming articles.

[13] Y. Sakakibara, Grammatical inference in bioinformatics, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1051-1062, 2005.

[14] D. E. Muller and P. E. Schupp, Groups, the theory of ends and context-free languages, *Journal of Computer and System Sciences*, vol. 26, no. 3, pp. 295-310, 1983.

[15] M. Plicka, J. Janousek, and B. Melichar, Subtree oracle pushdown automata for ranked and unranked ordered trees, in *Federated Conference on Computer Science and Information Systems (FedCSIS)*, Szczecin, Poland, 2011, pp. 903-906.

Mengxiao Qian is currently an undergraduate student in Lab. of Information Security, School of Computer Science, Huazhong University of Science and Technology. She is interested in information security.

Xingbo Xu is currently pursuing a master degree in Lab. of Information Security, Huazhong University of Science and Technology. He got his BS degree from Hubei University for Nationalities in 2013. His principal research interest is information security, network security, and malicious code.

Lansheng Han is currently an associate professor of Huazhong University of Science and Technology. He got his PhD degree from Huazhong University of Science and Technology in 2006. He has presided more than 15 projects, including two National Natural Science Foundations, and published more than 40 papers. His principal research interests are information security, malicious code, and connection models of network.

Cai Fu is currently an associate professor, deputy director of Information Security Institute of Computer School. He got his PhD degree from Huazhong University of Science and Technology in 2007. His main research interests include wireless networking security, routing algorithms, and distributed computing.

Kwisaba Hamza is currently pursuing a master degree at Huazhong University of Science and Technology, Wuhan, China. He got his BS degree from Makerere University, Uganda in 2011. His research interests are information security and network security.