I.J. Computer Network and Information Security, **2011, 3, 1-7**
Published Online April 2011 in MECS (http://www.mecs-press.org/)

A Novel Framework to Carry Out Cloud Penetration Test

Jianbin Hu
School of EECS, Peking University, Beijing, China
Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing, China
Email: hjbin@infosec.pku.edu.cn

Yonggang Wang, Cong Tang, Zhi Guan, Fengxian Ren, Zhong Chen
School of EECS, Peking University, Beijing, China
Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing, China
Email: {wangyg, tangcong, guanzhi, renfx, chen }@infosec.pku.edu.cn

***Abstract*—in current cloud services, users put their data and resources into the cloud so as to enjoy the on-demand high quality applications and services. Different from the conventional services, users in cloud services lose control of their data which is instead manipulated by the large-scale cloud. Therefore, cloud service providers (CSP) guarantee that the cloud which they provide is of high confidence in accuracy and integrity. Traditional penetration test is carried out manually and has low efficiency. In this paper, we propose FPTC, a novel framework of penetration test in cloud environment. In FPTC, there are managers, executors and toolkits. FPTC managers guide FPTC executors to gather information from the cloud environment, generate appropriate testing scenarios, run matched tools in the toolkit and collect test results to do evaluation. The capacity and quality of the toolkit is a key issue in FPTC. We develop a prototype in which FPTC is implemented and the experimental results show that FPTC is helpful to automatically carry out penetration test in cloud environment.**

***Index Terms*—Penetration test, cloud computing, high confidence, framework**

I. INTRODUCTION

In current cloud services, users put their data and resources into the cloud so as to enjoy the on-demand high quality applications and services. Different from the conventional services, users in cloud services lose control on their data which is instead manipulated by the large-scale cloud. Therefore, cloud service providers (CSP) should guarantee that the cloud which they provide is of high confidence in security. In a word, the problem that we focus on this paper is: **how to test whether a cloud is as secure as claimed by its CSP?**

In traditional network environment, we do penetration test to show the security status of the whole network. In other words, we collect several test tools and then execute them in sequence on some targeted hosts or networks. The execution of test tools and the evaluation of targeted hosts or networks are usually by hand and discrete. Here we give several important definitions: (1) “Penetration test” is “a method of evaluating the security of a computer system or network by simulating an attack from a malicious source” [1]. (2) A cloudlet (i.e., node/server, server cluster) is a minimum unit or component in cloud computing environment. (3) An unsecure cloudlet is a cloudlet which has vulnerabilities that can be exploited to do harm to the cloud environment that it resides.

Besides, because of the large scale of the cloud computing environment, it is increasingly likely that some cloudlets are accidentally miss configured or have been compromised as a result of un-patched security vulnerabilities. Cloudlets that span multiple administrative domains whose operators have different interests may face the threat of deliberate manipulation.

In this paper, we propose FPTC, a novel framework for penetration test in the cloud. In FPTC, there are managers, executors and toolkits. FPTC managers guide FPTC executors to gather information from the cloud environment, generate appropriate testing scenarios, run matched tools in the toolkit and collect test results to do evaluation. The capacity and quality of the toolkit is a key issue in FPTC. We develop a prototype in which FPTC is implemented and the experimental results show that FPTC is helpful to automatically carry out penetration test in cloud environment.

To the best of our knowledge, there is not any study, raising a framework similar with FPTC, on ensuring and testing the security status of the cloud. The contributions of this paper are as follows:

- We propose FPTC, and we are the first to propose a novel framework for penetration test in the cloud.
- We introduce cloud penetration test manager and cloud penetration test executor to manage and execute penetration tasks in FPTC.
- We analyze the performance of FPTC under several different circumstances of cloud computing, and present the effectiveness of FPTC on the security of the cloud.

Outline: The rest of this paper is organized as follows. We present the related work in Section II. Section **III** describes the design rationale of FPTC. Then, the evaluation is discussed in Section IV. Finally, we present conclusion and future work in Section V.

II. RELATED WORK

The study in [2] reports a survey about the security issues in the context of cloud storage services and the recent research on addressing these issues. Then, the study in [3] is a more general survey of cloud computing services. Both of these papers point out some of the same challenges that motivate our work. Previous work has shown how to apply responsibility to individual applications (e.g., [4], [5], [6], [7]). However, to the best of our knowledge, this paper is the first to propose responsibility for an entire platform.

The study of the Byzantine failure model originates in [8] and [9]. The study about state machine replication [10], [11] is a classic technique for masking a limited number of such Byzantine faults. Some previous Byzantine fault tolerance (BFT) protocols [12], [13], [14] can mask faults as long as less than one-third of the nodes are faulty [15]. The study in [16] presents the BAR model, which can tolerate a limited number of Byzantine nodes plus an unlimited number of "rational" nodes. Moreover, previous study in [17] described a protocol that hides the malicious influence of Byzantine faults by simulating more benign "identical Byzantine" faults on the top of them. Simulations of even more restrictive classes of omission and crash faults in the Byzantine failure model were proposed in [18], [19], [20], and [21]. They are typically designed for broadcast-based algorithms and assume a synchronous system or a large fraction of correct nodes.

In addition, trusted computing [22] is an alternative approach to achieving some of the guarantees we propose. Nevertheless, it typically requires trusting the correctness of large and complex codebases (e.g., hypervisors, device drivers, or entire kernels) which are still beyond the reach of state-of-the-art verification techniques. In contrast, some forms of responsibility have been implemented without special hardware and with very little trusted code. Other forms (e.g., responsibility for data confidentiality) may require some platform support, but we expect that small and simple primitives comparable to TrInc will be sufficient.

III. FPTC

Assuming that in FPTC the penetration test is carried out only from the CSP's perspective, we ignore penetration tests originated from users. There are two reasons: (1) users seldom have the capabilities to do integrate and accurate penetration test because of lack of corresponding knowledge and techniques; (2) there exist malicious users who may take occasion to destroy the cloud environment. In this section, we mainly introduce the components and structure of FPTC. First, we introduces the basic structure of FPTC in Section III-A. Then, we describe several main components in FPTC and their functionalities in Section III-B. Finally, we introduce the workflow of FPTC in Section III-C.

A. Structure

The main structure of FPTC is shown in Fig. 1. Obviously, there is a FPTC manager, several FPTC executors, and several FTPC toolkits. FPTC manager is in charge of the management of the whole framework for penetration test in the cloud. FPTC executor is the module which carries out the penetration test process practically. In FPTC toolkit, there are tools which are designed for penetration test in the cloud environment.

B. Main Components

1) FPTC Manager: FPTC manager is in charge of the management of the whole framework for penetration test in the cloud. Its responsibilities are as follows:

- Task input and output: receive the penetration test tasks from the user or other applications and return corresponding results.
- User interaction response: call appropriate modules to response to users' all kinds of interactions in the system according to the types of their actions.
- Task configuration: Different penetration test tasks require different configuration. The manager helps to assign the locations of FPTC executors to obtain high efficiency and accuracy.
- Executor Monitoring: monitor all the executors (to be introduced later) in the test environment to assure everything is OK. When there exist failed cloudlets or executors, the manager should reconfigure the task in time and report to applications or administrators in higher levels.
- Communication establishment: establish communication mechanism among modules in the whole framework, control the dataflow and message flow, make testers to know the progress of the test.

2) FPTC Executor: FPTC executor is the module which carries out the penetration test process practically. Its responsibilities are as follows:

- Organizing specific test: Extract and explain test scripts from other applications, and control the sequential execution of test tools in the toolkit (to be introduced later).
- Driving tool execution: receive the test commands of the manager, start the real execution of corresponding tools in the toolkit to do the penetration test, and return the test result to the manager.
- Validate test results: when the execution result of tools cannot be obtained, the FPTC executor is in charge of sending commands to some validation modules and return the validation result to the manager.

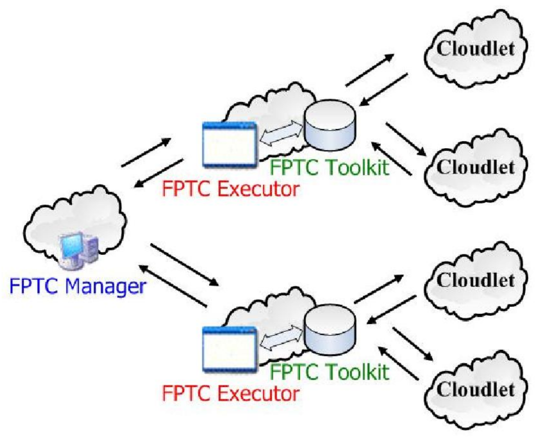

Fig. 1: Structure of FPTC

There are also several problems or key issues in FPTC Executors: (1) The parallel execution of tools from different FPTC executors, which means how to execute tools and collect execution results of different kinds; (2) The dealing with interrupt of execution which means how to do when the execution of a tool or a tool sequence suddenly stopped unexpectedly. (3) Different kinds of platforms or operating systems which makes the execution of tools more difficult because each tool has its own residing system or platform. During a penetration test, the test targets are various and the time is variable. Therefore, the correctness of matching the system and the tool is a key issue.

3) FPTC Toolkit: In FPTC toolkit, there are large quantities of tools which are designed or collected for penetration test in the network or cloud environment. There is also a database which stores the pre-conditions and post-conditions of the tools. The pre-conditions include user privileges, running services, vulnerabilities and system versions. The post-conditions include privilege promotion, vulnerability exploit, and knowledge acquisition (such as user account and password).

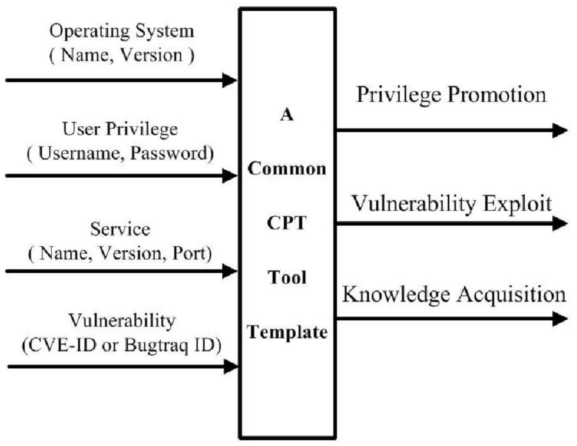

Fig. 2: Factors of a FPTC tool template

Each test tool in FPTC toolkit corresponds to one or more test rules in practice and is represented by a quintuplet (ID, Name, Para, Precond, Postcond). In this tuple, ID and Name represent identity and name of the test tool. Para represents the set of parameters of the tool when it is executed and it starts with abstract or default value. As test tools being executed, parameters become specific. Precond and Postcond respectively represent the precondition of using the test tool and the post-condition of using the tool. For better understanding, here we give a description example of a tool in FPTC toolkit. Tool *T* is represented by *T*=(2; username and password; privilege: admin; privilege: root), which means *T* is the No.2 tool in FPTC toolkit, *T*'s execution needs username and password, before *T*'s execution the privilege of the user should be no less than "admin" level, and the afterward privilege of using the tool *T* will be at least "root" level.

In order to make the testing result more comprehensive and accurate, the tools in the toolkit should be in large quantities and categories. And of course they should have been tested before they are integrated into the database. In FPTC, the degree of automation of tool management has an influence upon the speed of testing scenario construction and testing sequences processing. Fig. 2 shows the above factors of a FPTC tool template.

C. Workflow

The workflow of FPTC is as follows:

- Step 1–Test Preparation
  In this step, the FPTC manager and executors should be installed on some cloudlets which depend on the test environment and target. Normally, one FPTC manager and several executors in a cloud is enough. The FPTC manager sends messages to guide executors to carry out tasks. The executors run tools in the toolkit and collect corresponding results that will be returned to the FPTC manager.
- Step 2–Information Gathering
  The FPTC manager orders the FPTC executor to run scanners or sniffers to scan the whole cloud environment (i.e., all the cloudlets in the cloud) to gather information such as vulnerabilities, services and open ports. Under some circumstances, the information of the environment has been provided by CSP before the cloud penetration test which makes the workflow more simple and efficient. By this step, FPTC knows the cloud topology by obtaining answers to the following questions: How many cloudlets are there in the cloud? What vulnerabilities do they have? Are there any certain applications that have been installed on these cloudlets?
- Step 3–Scenario Construction
  Combining the information of the environment and the toolkit, FPTC constructs a testing scenario as the testing guide which will be carried out in Step 4. The testing scenario specifies the executing sequences of tools in the toolkit that are helpful to fulfill the test. The form of a testing scenario can be a test graph, a test tree or just test sequences. In a test graph, nodes represent certain states of a cloudlet and edges represent the tools which cause the transfer between states. The graph is generated by DFS or BFS algorithm. The test tree is similar to the test graph. However, the test sequences are just made up of a number of single state transfers. Each single state transfer can be represented as $<S_i\text{-}(T)\text{-}S_j>$, in which S_i and S_j respectively mean the start and end state of the transfer. *T* means the tool which causes the transfer.
- Step 4–Tool Execution
  No matter what representations are adopted, they can be transformed into execution sequences of tools, which indicate the orders of executing tools which have been prepared in the above steps to accomplish testing scenarios which have been constructed in Step 3. The objective of this step is to minimize the manual operations to raise testing efficiency. As to the execution of a single tool, FPTC doesn't need to focus on the detail of its execution. Instead, its execution can be seen as a

computing task that will be distributed in the cloud environment, the same as a normal cloud computing task. From this step it is obvious that FPTC is independent of cloud service delivery models (SaaS, PaaS and IaaS).

- Step 5–Testing Evaluation
  After collecting the execution results of all the selected tools in the toolkit, FPTC will do evaluation based on the comparison between constructed scenarios in Step 3 and execution results in Step 4. Then fill in the tables which may be designed according to some evaluation policies and generate reports that indicate some qualitative aspects of the testing goal. After that, an integrated process of penetration test in the cloud will be finished.

In general, the workflow of FPTC contains five steps which is concluded in Fig. 3: test preparation, information gathering, scenario construction, tool execution and testing evaluation. Obviously, there are strict orders between these steps. One step can be carried out if and only if the step before it has been done successfully.

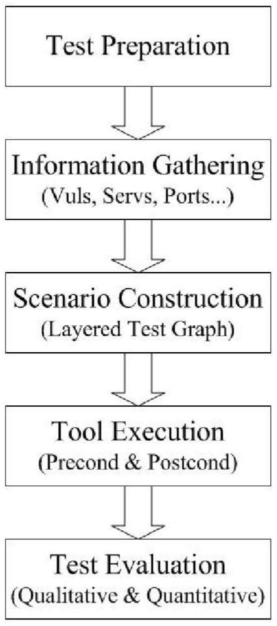

Fig. 3: Workflow of FPTC

IV. EVALUATION

We have described the structure, main components and workflow of FPTC so far. It is obvious that in a practical penetration test, FPTC needs to combine with the practical requirement to finish the penetration test task in time and also with high accuracy and efficiency. In this section, we first use an example to analyze FPTC's performance in penetration test in Section IV-A. Then we present several practical issues in constructing FPTC in Section IV-B. The details of test graph are introduced in Section IV-C.

A. Case Study

In this section, we present an example of using FPTC to fulfill a task of doing penetration test to a cloud environment. The target cloud environment was made up of 100 servers. Among these servers, we installed the FPTC manager in a server S_m. Several FPTC executors were installed in some of the rest servers, i.e., S_{ei} (i = 1, 2, …, n). Here we define the target servers as the servers that were neither S_m nor S_{ei} (i = 1, 2, …, n). We used the test graph as the representation of penetration test results. Before the evaluation, we didn't batch all the vulnerabilities in the target servers so that they had some vulnerabilities which could be exploited by some tools in the toolkit. After that, we used FPTC to fulfill a task of doing penetration test to this cloud environment. The experimental results are listed in Table I. In the table, we use N_t to denote the number of tools in the FPTC toolkit, n to denote the number of FPTC executors in the cloud environment, T to denote the time of the whole penetration test, N_a to denote the number of additional penetration achievement (measured by the number of unexpected services, open ports and vulnerabilities) besides predetermined vulnerabilities.

From the table above, we can see that the performance of FPTC which is measured by the number of additional penetration achievement increased when the number of FPTC executors or tools in the toolkit increased. The penetration time of T revealed the high efficiency of FPTC.

Table I: Case Study: Experimental results of FPTC

n	N_t	$T(min)$	N_a
1	50	8	131
1	100	14	232
2	50	21	127
2	100	39	207
5	50	37	109
5	100	67	184
10	100	113	171
10	50	60	103

B. Practical Issues

When FPTC is applied in real penetration test, there are some practical issues which are listed as follows:

- Elimination of redundant test sequences. By comparing existing test states and the tools that have used with further test states and tools to be used, the redundant test states in the test graph can be recognized and removed afterward. Therefore, the whole test scenario gets eliminated and has a smaller scale, which is helpful to fulfill certain penetration test tasks in less time.
- Low labor requirement. By providing remote control services such as RealVNC, FPTC supports that the administrator of the whole cloud environment doesn't need to walk among the cloudlets. What he needs to do is just to install several FPTC executors on some cloudlets and

manage them according to the FPTC manager which is installed in one cloudlet. Therefore, the administrator just sits in one cloudlet environment but keeps his eye on the executors and further the whole cloud.

- Capacity of toolkits. Theoretically, the larger the number of tools in the toolkits is, the better the effect of penetration test will be. However, more tools make it time-consuming for the penetration test. Therefore, the key issue related to FPTC toolkit is to maintain a tool database filled with tools of clear category, fewer cross preconditions, and high success ratios.

C. Test Graph

- Definition and introduction: A cloud penetration test graph (CPTG) is a graphically representative method to evaluate the results of cloud penetration test. In traditional CPTG, nodes represent test states and directed edges represent tools (or rules) that cause the transition between states.
- Layered CPTG: In large-scale cloud environment, because of large quantities of cloudlets, it usually takes a much longer time to construct the CPTG and the ultimate graph is always very large. Therefore, in practice, we adopt layered CPTG to do evaluation of cloud penetration test. During the construction and optimization of layered CPTG, CPTG is divided into “test supergraph” and “test subgraph” to make CPTG concision and to reduce complexity. For convenience, we briefly introduce the algorithms of the construction and optimization of layered CPTG in Table II, III and IV.

Table II: Construction Algorithm of test subgraph in layered CPTG

```
ConstructTestSubgraph(I, R)
Input: I (initial test state)
Input: R(the set of test rule)
Output: output_queue (the set of test states)
output_queue ß Φ;
state_queue ß state_queue+I ;
while(state_queue is not empty)
    cur_state ß the first state in state_queue;
    while(the test target has been given and reached)
        set cur_state as target state;
        delete the first state in state_queue;
        cur_state ß the first state in state_queue;
    for(each rule r in R that match cur_state)
        child_state ß cur_state;
        child_state ß child_state+the postcondition of r;
        if(cur_state!=child_state)
            if(child_state==a state having been constructed)
                set child_state as substate of cur_state;
            else
                set new state ID of child_state;
                set child_state as substate of cur_state;
            state_queue ß state_queue+child_state;
    output_queue ß output_queue+cur_state;
    delete the first state in state_queue;
return output_queue;
```

Table III: The algorithm of searching for the minimum test path set in constructing test supergraph in layered CPTG

```
FindMinimalTestPathSet(R)
Input：R (all the sets of test paths)
Output：MTPS (the minimal test path set)
MTPS ß Φ;
paths ß R;
while(true)
    minlen ß the step number of the shortest test path in paths;
    minpath ß the test path matched with minlen;
    MTPS ß MTPS+minpath;
    tmppaths ß paths－the test path which covers minpath;
    if (tmppaths is empty)
        return  MTPS;
    else
        paths ß tmppaths; //recursion
```

Table IV: The algorithm of searching for the minimal critical set of tests in constructing test supergraph in layered CPTG

```
FindMinimalCriticalTestSet(P,C)
Input：P (the set of test paths)
Input：C (the set of test subgraph ID)
Output：MCST (the minimal critical set of tests)
MCST ß Φ;
paths ß P;
while(paths is not empty)
    for(c∈unvc, the set of unvisited test subgraph ID)
        if(the number of test paths that cover c is the biggest)
            tmpaths ß the set of test paths that cover c;
            MCST ß MCST＋c;
            Unvc ß C－MCST;
    paths ß paths－tmpaths;
return MCST;
```

V. CONCLUSION AND FUTURE WORK

In current cloud services, users put their data and resources into the cloud so as to enjoy the on-demand high quality applications and services. Different from the conventional services, users in cloud services lose control on their data which is instead manipulated by the large-scale cloud. Therefore, cloud service providers (CSP) guarantee that the cloud which they provide is of high confidence in accuracy and integrity. Traditional penetration test is carried out manually and has low efficiency. In this paper, we propose FPTC, a novel framework of penetration test in cloud environment. In FPTC, there are managers, executors and toolkits. FPTC managers guide FPTC executors to gather information from the cloud environment, generate appropriate testing scenarios, run matched tools in the toolkit and collect test results to do evaluation. The capacity and quality of the toolkit is a key issue in FPTC. We develop a prototype in which FPTC is implemented and the experimental results show that FPTC is helpful to automatically carry out penetration test in cloud environment.

In the future, we plan to: (1) change the structure of FPTC according to various practical penetration test tasks; (2) enlarge the capacity of the FPTC toolkit to improve FPTC’s performance and integrity; (3) make the workflow of FPTC more automatic by better encapsulation of tools in FPTC toolkit.

ACKNOWLEDGMENT

This work was supported in part by the NSFC under grant No. 60773163, No. 60873238, No. 60970135 and No. 61003230, the National Significant Science and Technology Projects under grant No. 2009ZX01039-001-001, as well as the PKU PY Project under grant No. PKUPY2010-005.

REFERENCES

[1] "Penetration Test. http://en.wikipedia.org/wiki/Penetration test/."

[2] C. Cachin, I. Keidar, and A. Shraer, "Trusting the cloud," *SIGACT News*, vol. 40, no. 2, pp. 81–86, 2009.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," in *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28.*, 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[4] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: practical accountability for distributed systems," in *SOSP*, 2007, pp. 175–188.

[5] A. Haeberlen, I. C. Avramopoulos, J. Rexford, and P. Druschel, "Netreview: Detecting when interdomain routing goes wrong," in *NSDI*, 2009, pp. 437–452.

[6] N. Michalakis, R. Soule, and R. Grimm, "Ensuring content integrity for untrusted peer-to-peer content distribution networks," in *NSDI*, 2007.

[7] A. R. Yumerefendi and J. S. Chase, "Strong accountability for network storage," *TOS*, vol. 3, no. 3, 2007.

[8] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.

[9] M. C. Pease, R. E. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[10] L. Lamport, "Using time instead of timeout for fault-tolerant distributed systems," *ACM Trans. Program. Lang. Syst.*, vol. 6, no. 2, pp. 254–280, 1984.

[11] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, 1990.

[12] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, 2002.

[13] H. V. Ramasamy, A. Agbaria, and W. H. Sanders, "A parsimonious approach for obtaining resource-efficient and trustworthy execution," *IEEE Trans. Dependable Sec. Comput.*, vol. 4, no. 1, pp. 1–17, 2007.

[14] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, "Separating agreement from execution for byzantine fault tolerant services," in *SOSP*, 2003, pp. 253–267.

[15] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *J. ACM*, vol. 32, no. 4, pp. 824–840, 1985.

[16] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, "Bar fault tolerance for cooperative services," in *SOSP*, 2005, pp. 45–58.

[17] G. Bracha, "Asynchronous byzantine agreement protocols," *Inf. Comput.*, vol. 75, no. 2, pp. 130–143, 1987.

[18] T. K. Srikanth and S. Toueg, "Simulating authenticated broadcasts to derive simple fault-tolerant algorithms," *Distributed Computing*, vol. 2, no. 2, pp. 80–94, 1987.

[19] G. Neiger and S. Toueg, "Automatically increasing the fault-tolerance of distributed systems," in *PODC*, 1988, pp. 248–262.

[20] B. A. Coan, "A compiler that increases the fault tolerance of asynchronous protocols," *IEEE Trans. Computers*, vol. 37, no. 12, pp. 1541–1553, 1988.

[21] R. A. Bazzi and G. Neiger, "Simplifying fault-tolerance: providing the abstraction of crash failures," *J. ACM*, vol. 48, no. 3, pp. 499–554, 2001.

[22] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *HotCloud'09: Proceedings of the 2009 conference on Hot topics in cloud computing*. Berkeley, CA, USA: USENIX Association, 2009, pp. 3–3.

Jian-Bin Hu received his Ph.D. degree in 2002 from Computer Science Department of Peking University, where now he is an associate professor. His research interests include wireless sensor networks and embedded system design.

He has published lots of papers in international conferences and journals. He also won several Military Science and Technology Advancement Awards, including the second prize for twice and the third prize triple. He also has 2 patents of invention.

Yong-Gang Wang received his B.S. degree in 2004 from EECS of Peking University, where now he is a Ph.D. student. His research interests include security and privacy protection in online social networks.

He has published several papers in top-tier conferences, such as ICPADS, CISIS, FSKD, APWEB, and so on. He won Google Excellence Scholarship in 2010.

Cong Tang received his B.S. degree from National University of Defense Technology in 2005. He is currently a Ph.D. student at Peking University. His research interests include security and privacy in online social networks.

He was a visiting researcher in NYU between 2008 and 2009. He has published several papers in top-tier conferences, such as P2P, AINA, ICNS, and so on.

Zhi Guan received his B.S. degree from Dalian University of Technology and Ph.D. degree from the Peking University, in 2002 and 2009 respectively. He is currently an assistant professor at Peking University. His research interests include applied cryptography, security and privacy of RFID.

Feng-Xian Ren received her B.S. degree from Hunan University in 2009. She is currently a master student at Peking University. Her research interests include applied cryptography, security and privacy of ONS.

Zhong Chen is currently a professor of School of EECS, Peking University, and director of the Network and Information Security Research Group of the Software Institute. He received his B.S. and Ph.D. degrees from Peking University, in 1983 and 1988 respectively. He has wide interests in network and information security, system software and embedded system, SW/HW co-design methodology and domain-specific software engineering.

He is IEEE and Computer Society Member, Senior Member of Chinese Institute of Electronics since 1996, Managing Director of Directorate of China Software Industry Association since 1998, Co-chair of professional committee of Information Security and Privacy of China Computer Federation since 2002, Expert of Technical Auditing Committee of securities online trading, China Securities Regulatory Commission, and Vice-chairman of Editorial Committee of the Journal of Network Security Technologies and Application since 2001.