SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks (2013)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.754

RESEARCH ARTICLE

C^2Detector: a covert channel detection framework in cloud computing

Jingzheng Wu[1], Liping Ding[1], Yanjun Wu[1], Nasro Min-Allah[3], Samee U. Khan[4]* and Yongji Wang[1,2]

[1] National Engineering Research Center for Fundamental Software, Institute of Software, Beijing, China
[2] State Key Laboratory of Computer Sciences, Institute of Software, Beijing, China
[3] COMSATS Institute of Information Technology, Islamabad, Pakistan
[4] North Dakota State University, Fargo, ND 58108-6050, USA

ABSTRACT

Cloud computing is becoming increasingly popular because of the dynamic deployment of computing service. Another advantage of cloud is that data confidentiality is protected by the cloud provider with the virtualization technology. However, a covert channel can break the isolation of the virtualization platform and leak confidential information without letting it known by virtual machines. In this paper, the threat model of covert channels is analyzed. The channels are classified into three categories, and only the category that is new to cloud computing is concerned, for example, CPU load-based, cache-based, and shared memory-based covert channels. The covert channel scenario is modeled into an error-corrected four-state automaton, and two error-corrected algorithms are designed. A new detection framework termed C^2Detector is presented. C^2Detector includes a captor located in the hypervisor and a two-phase synthesis algorithm implemented as Markov and Bayesian detectors. A prototype of C^2Detector is implemented on Xen hypervisor, and its performance of detecting the covert channels is demonstrated. The experiment results show that C^2Detector can detect the three types of the covert channels with an acceptable false positive rate by using a pessimistic threshold. Moreover, C^2Detector is a plug-in framework and can be easily extended. It is believed that new covert channels can be detected by C^2Detector in the future. Copyright © 2013 John Wiley & Sons, Ltd.

KEYWORDS

covert channel detection; Xen; cloud computing; virtualization; Markov model; Bayesian model

***Correspondence**

Samee U. Khan, North Dakota State University, Fargo, ND 58108-6050, USA.
E-mail: samee.khan@ndsu.edu

1. INTRODUCTION

With the dynamic and elastic allocation of resource and service, cloud computing has become increasingly popular [1–3]. The fundamental mechanism of cloud is virtualization, allowing virtual machines (VMs) to instantiate stand-alone operating systems (OSs) on demand on the basis of a software layer called VM monitor (VMM) or *hypervisor* [4]. Although the virtualization technology provides strong isolation for the cloud, security and privacy are always the open problems [5]. Some of the problems are essentially traditional web application and data-hosting ones, for example, phishing, downtime, data loss, and password weakness. One of the new problems that are introduced by the shared environment to cloud computing is the covert channel (CC) attack [6]. By this way, information is leaked from cloud clients and, in the meanwhile, security provided by isolation breaks down [7,8].

To enhance the security of cloud computing, some protection mechanisms have been presented [9,10]. sHype [11] is a Mandatory Access Control-based security extension to Xen hypervisor [4]. For example, sHype enables the Chinese Wall and the Type Enforcement policy to specify whether or not the resources can be accessed by the VMs. Lares presents a hybrid approach, giving security tools the ability to monitor actively while still benefiting from the increased security of an isolated VM [12]. HyperSentry presents a novel framework to enable integrity measurement of a running hypervisor by introducing a software component [13]. However, these protection mechanisms may fail because the massive parallel cloud context creates numerous implicit high-resolution clocks used to construct the CCs [14].

Covert channel is a leakage mechanism used to transfer confidential information violating security policies specified by the information systems [15,16]. It is the main threat to

Copyright © 2013 John Wiley & Sons, Ltd.

the multi-level secure systems, for example, OSs [17,18], database systems [19], network [20,21], and cloud computing [7]. Trusted computer system evaluation criteria [15] and Common Criteria for Information Technology Security Evaluation (CC) [22] secure criterions require CC analysis when building secure systems. The objectives of CC analysis are identification, estimation capacity, detection, and handling [23,17,24,25]. This paper concerns the detection technology and presents a new detection framework C²Detector.

Detection in this paper is to determine the CCs in operational track records. Cabuk *et al*. [26,27] present an algorithm to detect TCP/IP network covert timing channel. They believe that the regularity of the packet intervals indicates the difference between the covert and normal channels. Therefore, they present two methods to measure the regularity. The first method examines whether the variance of the intervals remains constant using standard deviation. The second method computes the relative difference between each pair of the sorted intervals. Similar to that of Cabuk *et al*., Berk *et al*. [28] present a statistical algorithm to detect the CC whose packet intervals center around two different values. The algorithm compares the ratio of the mean packet count to the maximum packet count for normal traffic. The lower the ratio is, the higher the probability of having a CC hidden in packet intervals turns out to be. Nagatou *et al*. [29] define security automata and show the enforced properties. To detect several CCs at run time, they use an extra structure to emulate the behavior of a system by running a subsequence from an interleaved state sequence of processes. However, these detection methods and some other ones (e.g., [30,20]) are limited to the particular network or OSs, unsuitable for detecting the CCs in cloud computing because of the different formation mechanisms.

The CCs in cloud are induced by the massive shared computing resources, and they cannot be detected by the aforementioned methods. In this paper, the threat model of CCs is first analyzed, and the channels are classified into three categories. Only the channels between VMs new to cloud are considered, and three typical scenarios (e.g., CPU load-based, cache-based, shared memory-based channels) are demonstrated and their features are studied in depth. A detection framework named C²Detector is presented following some basic requirements: small modification to the protected cloud systems, flexible extension to detect new channels, and acceptable performance impact. An error-corrected four-state automaton is proposed to model the channel scenario, and a two-phase synthesis algorithm is designed to detect the CCs using Markov and Bayesian models. The prototype of C²Detector is implemented on Xen hypervisor, and the performance to detect the three typical CCs is investigated. The experiment results show that C²Detector is competent to detect these channels, and it is believed that C²Detector will be able to detect incoming CCs in the future by simple extension.

The distinguished contributions made in this paper are as follows:

- For the first time, the CCs in cloud computing are classified into three categories, and only the channels new to cloud are concerned.
- The CC scenario in cloud computing is modeled into an error-corrected four-state automaton, which is the basis of the detection scheme.
- A flexible framework named C²Detector is designed, consisting of a captor in hypervisor and a synthesis analyzer in the virtual domain. The performance to detect the three typical CCs is evaluated.
- A two-phase synthesis detection algorithm using Markov and Bayesian models is presented, and the pessimistic threshold is adopted to lower the false negative.

The rest of the paper is organized as follows. Section 2 discusses the threat model, typical scenarios, and some assumptions. Section 3 describes the design of C²Detector by introducing the challenges and the formal requirements, and describes the details of the two-phase synthesis algorithm. Section 4 describes the prototype implementation and evaluates the detection performance of C²Detector framework. Section 5 discusses the extendibility of C²Detector. Section 6 describes the related work, and Section 7 concludes this work.

2. THREAT MODEL, SCENARIOS, AND ASSUMPTIONS

In this section, the CC threat model and scenarios to the cloud computing are investigated, especially to the lifetime of VMs. The section ends with some assumptions used to design the CC detection framework.

2.1. Threat model

Cloud clients instantiate stand-alone OSs on demand, and deploy software or offer service to the application layer. When the tasks have finished, the VMs are destroyed. Some threats are presented in the lifetime of VMs as shown in Figure 1.

Start/stop attacks (A3,A4). Cloud platform allows users to start and deploy malicious hosts easily, which are used as distributed denial-of-service and Botnet attacks. In A3 attack, hackers tamper with the image of

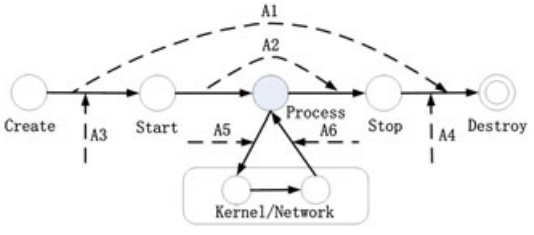

Figure 1. Lifetime threats to cloud computing, including tamper, leakage, and application attacks.

VMs and trojan in the system before startup. In A4 attacks, hackers tamper with the stored data to leak or steal the confidential data. A3 and A4 are induced by VMs, which lead to inside attacks becoming outside attacks.

Run-time attacks (A1,A2). Virtualization is the fundamental mechanism of the cloud, providing the abstract services of hardware resources to the OSs in VMs. A1 denotes the information leakage attack to inter-VMs, for example, the CPU load-based and cache-based CC. Although the hypervisor allocates virtual CPU (vCPU) to each VM, the tasks will run in the physical CPU eventually inducing A1 attacks. A2 denotes the CCs in VMs, for example, the event-flag channels in Linux [17,24]. A1 and A2 leak confidential information in or between VMs and cannot be eliminated by the deployed access control policies.

Application attacks (A5,A6). A5 attacks target and modifies the context information of OS calls. A6 attacks alter the function responses by intercepting and modifying the return values. In network, A5 and A6 may lead man-in-the-middle attacks. However, A5 and A6 are traditional attacks, which have been studied in depth [23].

2.2. Threat categories of covert channel

The aforementioned attacks cover the lifetime threats of cloud, and the bottom-up attack targets are hypervisor, VMs, and applications. In this paper, only the CC attacks (A1 and A2) are concerned. They can be classified into three categories as shown in Figure 2.

Intra-VM covert channels (CC1), that is, processes level CCs. Malicious processes P_i and P_j with different secure levels are located in the same OS in domain unit (DomU). P_i with the higher secure level leaks confidential information to the lower secure level P_j using the CC. However, the threat of the CC is limited to the stand-alone OS. Channels of CC1 have been studied for years, and some mature analysis methods can be referred to [17,23].

Cross-platform covert channels (CC2), that is, network level CCs. Malicious processes P_k and P_x are located in different domains and different hardware platforms. P_k and P_x communicate with each other through the network; therefore, the confidential information can be transmitted by the network storage and timing channels. Channels of CC2 are based on the network, which have been studied since 1987 [20,31].

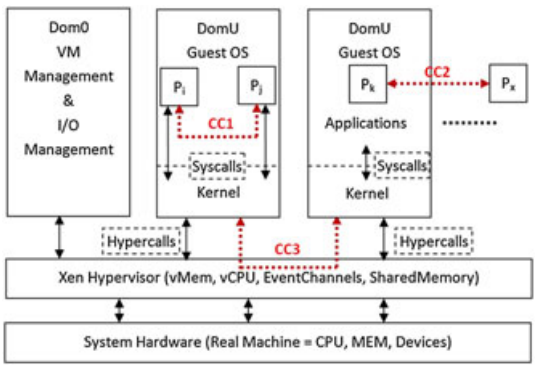

Figure 2. Three categories of covert channels in Xen virtual machines.

Inter-VM covert channels (CC3), that is, OS level CCs. Malicious processes are located in different domains but the same hardware platform. CC3 is introduced by the shared resource managed by the hypervisor [7,32,33]. Confidential information may be leaked by CC3 among competitive companies that is located in the same hardware, which will bring huge economic losses.

2.3. Threat scenarios of covert channel

Although there are a number of avenues to extract confidential information from cloud computing, the CC attack is more advanced. Only CC3 channels are new CCs to cloud, and the other two types are OSs and network CCs. In this paper, three typical CC scenarios belonging to CC3 are described.

CPU load-based channels [7,34]. It has been considered that any physical resources that are multiplexed between the attacker and target host may form a potential CC in VMs. One of the most common resources is CPU load, which can be approximated by the amount of time taken for certain computations. The confidential information is pre-encoded into a binary sequence. The sender and receiver transfer information by changing and observing the CPU load according to a certain communication protocol, for example, long waiting time to complete a task means bit 1 is transmitted, otherwise bit 0 is transmitted.

Cache-based channels [7,35]. The cached-based CC takes the different cache access latencies as the different bits. The sender uses the idle ones as transmitting bit 0 and the frantic accesses to memory block as transmitting bit 1. The receiver accesses a memory block of her own and observes the access latencies. High latency denotes the sender is evicting the receiver's data from the caches and means bit 1 is transmitted, otherwise bit 0 is transmitted.

Shared memory-based channels [32]. The shared memory-based channel takes different memory access intervals as the different bits. The sender sends covert messages by controlling the data sending time, and the receiver obtains the message by observing the data arrival time. The confidential information is encoded into the different intervals. For example, longer and shorter intervals denote bit 1 and bit 0, respectively. This type of CC is named shared-memory covert timing channel (SMCTC).

2.4. Assumptions

It is usually considered that the cloud computing is based on Xen virtualization platform consisting of the hardware, hypervisor, management domain, and some guest domains [4]. C²Detector framework runs on Xen platform. It assumes that only three types of CCs are concerned, which are CPU load-based, cache-based, and shared memory-based channel. It is also assumed that the CCs exist in the malicious domains along with some other innocent domains in the same cloud platform.

3. OVERVIEW OF C²DETECTOR

In this section, the design of C²Detector is discussed, starting off by describing the challenges. To guarantee the detection of the three types of channels, the requirements are formally stated. Then, C²Detector is investigated in detail, requiring the CC to be modeled into an error-corrected four-state automaton. Finally, a two-phase synthesis algorithm to detect the CCs using Markov and Bayesian models is presented.

3.1. Challenges

The first issue faced is that the traditional detection technology cannot be used in cloud computing. Although CC has been studied for almost 40 years, the researches mainly focus on OSs, database systems, and networks [23]. CC in cloud platform is a new topic, whose features have not been fully understood. In this paper, this new type of CC is modeled into an error-corrected four-state automaton, and the features are investigated.

The next issue is whether the framework can be flexibly extended to detect the incoming CCs. C²Detector addressed this problem by adopting a plug-in mechanism. To detect a new CC, the first step is to extract the features of the channel, for example, finding the shared resource and the values or states that can be used by the channels. Then, these features of the channel are normalized into detection parameters. Finally, an Extensible Markup Language (XML) configuration file containing these parameters is automatically generated and loaded by C²Detector as a plug-in.

The last issue is how to solve the bypass operations. Some operations that have super privileges cannot be captured in the traditional OSs, which may induce CCs and hence leads to the failure of detection. However, hypervisor or VMM in virtualization platform has a higher privilege than the OSs, where C²Detector resides in and captures all the operation tracks.

3.2. Formal requirements

C²Detector is a virtualization-based framework designed to detect the CC3 type of cover channels by monitoring the operations in VMs. To counter the aforementioned challenges, the design of C²Detector is based on the following four high-level formal requirements:

- Complete detection. Detection is to determine the real CC in operation records. All of the operations may be potential CCs, so they should be totally recorded and detected.
- Flexibility of extension. A well-functioning detector should have the flexibility to support both detecting new CCs and adding new detection algorithms.
- Acceptable performance impact. An additional performance impact introduced by C²Detector should be within acceptable limits. This requirement ensures that the overall performance of the virtualization platform should not be hurt. It also requires that C²Detector should detect the CCs in real time. Once any CC is detected, the alarm will be issued.
- Anonymous detection. Operations of the VMs should be protected in privacy. C²Detector only investigates the features of shared resources, for example, vCPUs, cache and shared memory. The precise operations will not be inferred.

3.3. C²Detector framework summary

Figure 3 shows the architecture of C²Detector. C²Detector consists of two-part components. One is a core active captor, locating in hypervisor and capturing all the operations triggered by the guest OSs. The other one is a back-end detector locating in Dom0, used to analyze the captured operation records and detect the CCs from them.

The captor places a hook inside the hypervisor and monitors all the hypercalls trigged by VMs. A hypercall is conceptually similar to a system call. The hypercall interface allows domains to trigger a synchronous software trap into hypervisor to execute a privileged operation, and the communication from hypervisor to a domain is provided through an asynchronous event mechanism [36]. Hypervisor can intercept any instructions that change the states of the machine in a way that impacts other processes. Therefore, it is an ideal monitor place where any hypercalls cannot bypass. In CC3 channels, the malicious processes locating in different VMs cooperate with the shared resources to communicate indirectly. All the operations and each state of the shared resource will be recorded by the capture application. The records will be sent to the detector locating in Dom0.

The detector is the module to detect in C²Detector, which is designed as a plug-in form. The detector locates in Dom0, which has elevated privileges and manages the other domains. The detector first processes the record stream, normalizes the data, and finally determines CCs from the data.

The record stream is processed and saved concurrently by the captor. Then, the records are normalized. For example, the captor only records the operation times in shared memory-based channels. However, the real shared

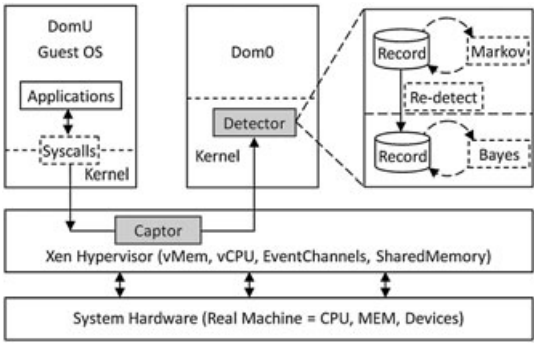

Figure 3. High-level view of C²Detector framework.

resource is the time intervals, so the intervals are calculated from the records. When the data have been normalized, it will be detected by the two-phase synthesis algorithm.

The detector is made up of two main blocks presently: Markov detection module and Bayesian detection module. A two-phase synthesis algorithm is implemented in both modules sequentially, which can be extended through plug-in in the future. The records first flow into the Markov detector; all the CCs will be detected with some false positives because of the pessimistic threshold. Then, the results flow into the Bayesian detector to refine. When the records have been detected by the detector, the outputs are the CCs.

The two-phase synthesis algorithm is designed according to the CC model, that is, an error-corrected four-state automaton. The automata model and the details of the algorithms will be introduced in the following.

3.4. Covert channel modeling

A CC consists of a shared resource, a sender process, and a receiver process. The sender transmits confidential information by changing the properties of the shared resource, and the receiver receives the message by viewing the changes. In CPU load-based, cache-based, and shared memory-based channels, the properties are the CPU load, cache access time, and the memory write intervals. The changes of these resource properties are in certain regular patterns. An error-corrected four-state automaton is designed to model the patterns [37,38].

Definition 1. *Error-corrected four-state automaton*
Error-corrected four-state automaton is a five-tuple

$$(Q, \Sigma, \delta, q, F)$$

which is shown in Figure 4, and the details of the tuple are described as follows.

- Q is a states set including the shared resources properties.
- Σ is an actions set abstracted from events involved in a system.

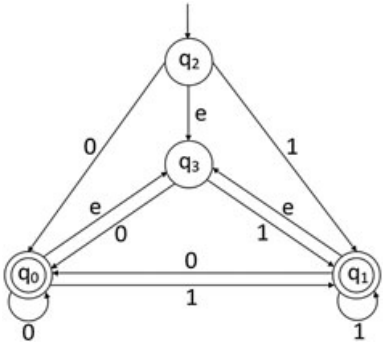

Figure 4. Error-corrected four-state automaton model.

- $\delta : Q \times \Sigma \to Q$ is a transition functions set denoting that an action is triggered from one state to another.
- $q \in Q$ is the initial state.
- $F \subseteq Q$ is the final states set.

Taking shared memory-based channel for example, the property of the channel is the memory write intervals [32]. The longer interval denotes bit 1, and shorter one denotes bit 0. SMCTC has only four states in the transmission process as shown in Figure 4, and all the different states belong to set Q. Set δ includes all the transition functions.

At the beginning of the transmission cycle, the initial state is q_2. If bit 0 is sent, the state changes from q_2 to q_0, expressed as $q_2 \to q_0$. If bit 1 is sent, the state changes from q_2 to q_1, expressed as $q_2 \to q_1$. When an error occurs, the state changes from q_2 to q_3, expressed as $q_2 \to q_3$. When an error occurs, the state changes to q_0 or q_1 according to the error-corrected algorithm.

However, where do the errors come from? In SMCTC, the memory write intervals are not exactly consistent. For example, a sequence of intervals captured is expressed as $T = \{t_0, t_1, \ldots, t_n\}$. We take ΔT_l, with $t_a < \Delta T_l < t_b$, as long interval and ΔT_s, with $t_c < \Delta T_s < t_d$, as short interval, $\Delta T_l > \Delta T_s$. Here, t_a, t_b, t_c, t_d mean the ranges of the intervals. If an interval t_i is out of the ΔT_l and ΔT_s, an error occurs. The reason for this is that some interferences are running, and the transition are $q_2 \to q_3$, $q_0 \to q_3$, and $q_1 \to q_3$.

To deal with the errors, two simple error-corrected algorithms are presented in this paper.

- Value closer-based error-corrected algorithm. This algorithm is based on the approximation. Take t_i as an example, if $|t_i - \Delta T_l| < |t_i - \Delta T_s|$, it is considered that bit 1 is transmitted, and vice versa.
- Probability-based error-corrected algorithm. This algorithm is based on the predetermined probability. If an error occurs, it is believed that bit 1 is transmitted with the probability of t and bit 0 with probability of $(1 - t)$.

The first algorithm is easy to implement, and the second one needs to predetermine the probability. The Markov detector and the Bayesian detector adopt the second algorithm in this paper. The repeated errors may mean the channel environment has changed, and the values of ΔT_l and ΔT_s should change correspondingly. The dynamic adjustment is complex, which is not considered in this paper.

Therefore, the automata model of SMCTC is instantiated as follows (Figure 4).

- $Q = \{q_0, q_1, q_2, q_3\}$ contains the four states of the shared resource property.
- $\Sigma = \{0, 1, e\}$ contains all the terminators normalized from the intervals captured.
- $\delta : Q \times \Sigma \to Q$ is a set of transition functions expressed as a state transition matrix as follows.

	0	1	*e*
q_0	q_0	q_1	q_3
q_1	q_0	q_1	q_3
q_2	q_0	q_1	q_3
q_3	x	y	ϕ

Triggers 0, 1, and e are in the first line of the matrix, and the current states are in the first row. The elements in the matrix are the next states transiting from the current states. For example, q_0 in the second line and the second row denotes the state changed from q_0 when bit 0 is transmitted, expressed as $q_0 \rightarrow q_0$.

In the last line of the transition matrix, the states transiting from q_3 depend on the error-corrected algorithms expressed by x and y. When an error occurs, the algorithm determines whether a bit 0 or 1 is transmitted and the corresponding state is q_0 or q_1. The state will not stay at p_3, denoted by φ in the matrix.

- q_2 is the initial state.
- $F = \{q_0, q_1\}$ contains all the final states.

Scenarios of CPU load-based and cache-based channels can also be modeled into this error-corrected four-state automaton. The channel is assumed to take the binary encoding mechanism. If a multiple encoding mechanism is adopted, the corresponding automata just have more states. It will be discussed in Section 5.

3.5. Two-phase synthesis detection algorithm

The error-corrected four-state automaton describes the channels. The real shared resources are operation intervals, which cannot be stored by the channel, meaning memoryless. The next state in the automata just relates to the current state, which is triggered by the stochastic bit. According to these properties, a two-phase synthesis detection algorithm is presented to detect the CCs.

The algorithm is synthesized by Markov detection algorithm and Bayesian detection algorithm. If the change pattern of the shared resource properties is closer to the Markov model, it is believed that the sequence of the operations is transferring confidential information through some CC. The normal operation sequences are modeled into Bayesian model. If an operation sequence deviates from the Bayesian model, it is believed that a CC occurs. Markov and Bayesian models in the two-phase synthesis algorithm are complementary. Markov detector detects the CCs, and Bayesian detector distinguishes the CCs from the normal operation sequences.

3.5.1. Markov detection algorithm

Markov property refers to the memoryless property of a stochastic process [39–41]. CC model is such a stochastic process that the next state depends only upon the current state and has nothing to do with the previous status.

The error-corrected four-state automaton has only two final states q_0 and q_1. State q_2 in Figure 4 is the start state, and no other states can transform to it. The automata will not stay at q_3 because of the error-corrected algorithms. To simplify the model, q_2 and q_3 are pruned, and a two-state Markov model is presented in Figure 5. The trigger actions in the automata are stochastic, and the next state depends only upon the current state.

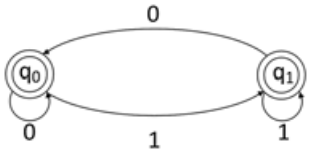

Figure 5. Two-state Markov model of covert channel.

The distribution probabilities of the triggers 0, 1, and e are set as p, q, and r, and $p+q+r=1$. Therefore, the probability of transition $q_2 \rightarrow q_0$ is p, the probability of $q_2 \rightarrow q_1$ is q, and the probability of $q_2 \rightarrow q_3$ is r. According to the second error-corrected algorithm, the state q_3 transits to q_0 and q_1 with probability t and $1-t$. The Markov model is instantiated as follows.

- State space. The Markov chain is $H(t), t = 1, 2, \ldots$, and the state space is $\Phi = \{q_0, q_1\}$ denoting the confidential bits transmitting. The number of the states in this model is N, and $N = 2$.
- State transition probability distribution is shown as

$$A = [a_{ij}]_{N \times N} = \begin{bmatrix} p + rt & q + r(1-t) \\ p + rt & q + r(1-t) \end{bmatrix}$$

In the matrix, a_{00} denotes the probability of transition from state q_0 to q_0, where the error state is hidden. The reason for $a_{00} = p + rt$ is that the distribution probability of bit 1 is p; the error is r and transmits to bit 1 with probability of t. All the other elements are calculated in the same way.

- Observable symbols and distribution probability. The observation symbols correspond to the records modeled. The individual symbols are denoted as

$$V = \{v_0, v_1, \ldots, v_M\}$$

The observation symbol probability distribution in state i is

where $$B = \{b_i(k)\}$$

$$b_i(k) = p[(v_k \text{ at } t) | q_t = S_i], 1 \le i \le N, 1 \le k \le M$$

$b_i(k)$ means the appearance probability of action v_k at time t under the state S_i. M is the number of distinct observation symbols per state. Therefore, values of B in the CC is

$$b_0(0) = p + rt, \quad b_0(1) = 0,$$
$$b_1(0) = 0, \quad b_1(1) = q + r(1-t)$$

- Initial state distribution. This distribution is $\pi = \{\pi_0, \pi_1\}$, where $\pi = \{p + rt, q + r(1-t)\}$ in this model. Values of π is calculated with the distribution probability of the bits of 0, 1, and e when initializing.
- Observation sequence is

$$O(t), t = 1, 2, \ldots, T, O(t) \in V$$

where T is the total number of the observed signals.

The CC is described by the Markov model as follows,

$$\lambda = (A, B, \pi)$$

and the probability of the observation sequence $O = \{O_1, O_2, \ldots\}$ is calculated. Given the model λ, this probability is defined as $P(O|\lambda)$. $P(O|\lambda)$ denotes the observation sequence built from the model of λ, if the value is bigger enough, it is believed that a CC occurs.

To calculate $P(O|\lambda)$, the forward variable

$$a_t(i) = P\{O_1 O_2 \cdots O_t, H(t) = S_i|\lambda\}$$

must be considered first, and it is the probability of the partial observation sequence $(O_1 O_2 \cdots O_t)$ until time t and the state is S_i. $a_t(i)$ is calculated inductively as follows [39]:

- Initialization:

$$a_1(i) = \pi_i b_i(O_1), 1 \le i \le N$$

- Induction:

$$a_{t+1}(i) = \left[\sum_{j=1}^{N} a_t(i) a_{ij}\right] b_j(O_{t+1}), \quad 1 \le t \le T-1, \quad 1 \le j \le N$$

- Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} a_T(i)$$

On the basis of the Markov model $\lambda = (A, B, \pi)$, $P(O|\lambda)$ is calculated as follows.

$$\begin{aligned} P(O|\lambda) &= \sum_{i=1}^{N} a_T(i) = a_T(0) + a_T(1) \\ &= \left[\sum_{i=1}^{N} a_t(0) a_{i0}\right] b_0(O_{t+1}) + \left[\sum_{i=1}^{N} a_t(1) a_{i1}\right] b_1(O_{t+1}) \\ &= [a_t(0) a_{00} + a_t(1) a_{10}] b_0(O_{t+1}) \\ &\quad + [a_t(0) a_{01} + a_t(1) a_{11}] b_1(O_{t+1}) \\ &= (p + rt)[a_t(0) + a_t(1)] b_0(O_{t+1}) \\ &\quad + (q + r(1 - rt))[a_t(0) + a_t(1)] b_1(O_{t+1}) \end{aligned}$$

Therefore, the following equation is obtained directly,

$$\begin{aligned} &a_{t+1}(0) + a_{t+1}(1) = [a_t(0) + a_t(1)] \\ &\cdot [(p + rt) b_0(O_{t+1}) + (q + r(1-t)) b_1(O_{t+1})] \end{aligned}$$

Finally, $P(O|\lambda)$ is obtained

$$P(O|\lambda) = \prod_{i=1}^{T} [(p + rt) b_0(O_i) + (q + r(1-t)) b_1(O_i)]$$

where p, q, and r are the signal probabilities and $p + q + r = 1$.

The inputs of the Markov detector are the captured records, and the outputs are whether the records are CCs or not. $P(O|\lambda)$ is the decision factor and calculated from the records. A pessimistic threshold is introduced to set the resulting boundary.

Definition 2. *Pessimistic threshold*

Pessimistic threshold is calculated in the worst case. With this value, there are no false negative results, but some false positive ones. P_{thr} is used to denote the pessimistic threshold.

The Markov detector takes the situation that there is no CC as the worst case, so the Pessimistic Threshold is calculated as follows.

- The first step of calculating the pessimistic threshold is building a test bed, where all the operations are normal without CCs.
- Then, the value is calculated by Markov detector under this situation, which is relative bigger.
- Thirdly, the operations in VMs is executed repeatedly, and all the values are calculated and recorded.
- Finally, the smallest values is adopted and set as the pessimistic threshold.

When the pessimistic threshold is calculated, the decision policy of Markov detector is

$$P(O|\lambda) < P_{\text{thr}}$$

If $P(O|\lambda)$ of a record is smaller than P_{thr}, it is considered as a CC. Because the P_{thr} is calculated in the worst case, some normal records may be mistaken as potential CCs. Therefore, a Bayesian detector is needed to refine the results and lower the false positive.

3.5.2. Bayesian detection algorithm

Bayesian reasoning provides a probabilistic approach to inference. It is based on the assumption that the quantities of interest are governed by probability distributions and that the optimal decisions are made by reasoning about these probabilities together with observed data [42–44]. Naive Bayesian classifier is a highly practical Bayesian learning method whose performance is comparable with a neural network and decision tree learning in some domains.

In this paper, a naive Bayesian detector is designed and classifies the captured records into two classes, the CCs and the normal operations. Each task processed by the Bayesian detector is described by a conjunction of attribute values, for example, $x = < a_1, a_2, \ldots, a_n >$ (n is the number of the properties). The input of the detector is the records detected by Markov detector, including some false negative records. The output of the detector is a target function $f(x)$ whose value domain is $V = \{yes, no\}$. Values of *yes* and *no* indicate whether x is a CC or not. To the detector, the training samples are $X = \{x_1, x_2, \ldots, x_n\}$. Both the CCs and normal samples are used to train the detector, respectively. Then, the most probable target value v_{MAP} is calculated as

$$v_{\mathrm{MAP}} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2, \ldots, a_n)$$

where $P(v_j|a_1, a_2, \ldots, a_n)$ denotes the probability to classify a certain sequence $x = \langle a_1, a_2, \ldots, a_n \rangle$ into class v_j.

Then, v_{MAP} is rewritten into the following expression as

$$v_{\mathrm{MAP}} = \operatorname{argmax}_{v_j \in V} \frac{P(a_1, a_2, \ldots, a_n | v_j) P(v_j)}{P(a_1, , a_2, \ldots, , a_n)}$$

$$= \operatorname{argmax}_{v_j \in V} P(a_1, a_2, \ldots, a_n | v_j) P(v_j)$$

where $P(a_1, a_2, \ldots, a_n)$ is removed because it is a constant independent of v_j.

Bayesian detector is based on the simple assumption that the target attribute values are conditionally independent. Therefore, the probability of the conjunction $a_1, a_2, \ldots, a_n$ is calculated as follows.

$$P(a_1, a_2, \ldots, a_n | v_j) P(v_j) = \prod_{i=1}^{n} P(a_i | v_j)$$

Finally, the Bayesian detector is expressed as

$$v_{\mathrm{BD}} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_{i=1}^{n} P(a_i | v_j)$$

where v_{BD} is the target value of the Bayesian detector.

All the captured records are detected by the two-phase synthesis algorithm. Both of the algorithms are easy to be implemented and applied online. The records are detected by Markov and Bayesian detectors sequentially, and the final outputs are the detection results.

4. IMPLEMENTATION AND EVALUATION

This section describes the implementation and evaluation of C²Detector framework on Xen hypervisor. Firstly, the three types of CCs concerned in this paper are introduced. Secondly, the captor in hypervisor and the detector with the two-phase synthesis detection algorithm in Dom0 are described. Then, the configurations of the CCs are demonstrated, and the operational records are captured. At last, the performance of the captor, the Markov detector, and the Bayesian detector in C²Detector is evaluated.

4.1. Covert channels scenarios

A transmission cycle of a CC is shown in Figure 6 [45]. The confidential information is transmitted from the sender to the receiver. The sender encodes the information into binary bits at first. Then, the properties of the shared resources are changed by the sender according to the bits. Finally, the receiver observes the changes and decodes the confidential information from these changes. The sender and receiver predetermine the parameters (e.g., decoding mechanism) and repeat the process until all the confidential information has been transmitted.

CPU load-based, cache-based, and shared memory-based channels are implemented according to the transmission cycle. The shared resources of the channels are CPU load, cache access time, and the memory writing intervals.

CPU load-based CCs use CPU load rate to denote the confidential information [7,34]. A web server is running in the receiver VM. The sender encodes the confidential information into binary bits and issues numerous HTTP requests via *JMeter 2.4* (a utility for load testing HTTP servers). The receiver monitors the CPU utilization of the web server and decodes the confidential information from the changes.

The shared resource in cache-based CC is the cache access time [7,35]. The sender shares a common cache with the receiver and occupies the cache according to the encoded binary bits. The receiver observes the cache access time and decodes the confidential information from the time. A Prime–Probe protocol is used in the transmission cycle [46,47]. A basic construction of the Prime–Probe protocol is described as follows.

- PRIME: The receiver fills an entire cache set S by reading memory region M from memory space.
- IDLE: The receiver waits for a pre-specified Prime–Probe interval while the cache is being utilized by the sender.
- PROBE: The receiver times the reading of the same memory region M to learn the sender's cache activity on cache set S.

The shared resource in shared memory-based channel is the memory writing interval [32]. A memory sharing

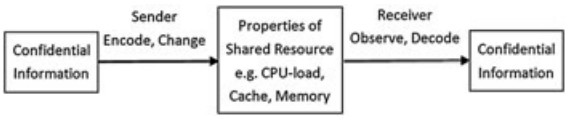

Figure 6. Transmission cycle of a covert channel scenario.

module is loaded into the guest OS. The sender encodes the confidential information and writes memory according to the different bits. The receiver observes the writing intervals and decodes the confidential information from them. The intervals may be affected by the VM loads, which will be investigated in the experiments.

4.2. Captor and detector

C²Detector framework is implemented in a desktop computer with an Intel® Core™2 Quad Q9400 2.66-Hz CPU, 320-GB disk, and 4-GB main memory. Xen hypervisor is 4.0.0, and each VM is allocated 512MB virtual memory running Fedora 8 Linux with kernel 2.6.34.1. The three types of CC programs used to leak the confidential information are implemented in C-Language.

The captor places the hook function *do_capture_init* in the hypervisor. *do_capture_init* first initializes a buffer *capture_buf* and then calls the function *do_capture_op* to record all the operational events. When a hypercall is triggered, *do_capture_op* records the event information, including *vcpu_id*, *dom_id*, *grant_ref_t*, and memory writing time. The information recorded is filled into *capture_buf* and sent to the detector in Dom0 at last. This framework is similar to Lares [12], which mainly focuses on memory integrity protections. Comparing with Lares, the captor monitors the operations related to the shared resources.

The detector is implemented as a Linux kernel module (LKM), including the Markov and Bayesian detectors. Dom0 is the privileged domain. Therefore, the LKM is automatically loaded into Dom0 when C²Detector starts. LKM is a module running in the kernel space, which has a small impact on the hypervisor performance. The Markov and Bayesian detectors are implemented in C-Language. C²Detector is easy to be extended to support other detection algorithms, for example, the decision tree and neural network algorithms. The inputs of the detector are the records from the captor, and the outputs are the detection results. If the captured record set is too large to be processed in a single VM online, the detector can extend to multi-VMs and process the records in parallel. In this paper, the parallel process is out of consideration.

4.3. Experimental settings

The confidential information transmitted from the sender to the receiver is 50 letters text. The sender and receiver are located in the different VMs running Fedora 8 Linux and use the same encode scheme. The text is encoded into 400-bit-length binary string. The transmission cycle is shown in Figure 6. The different bits denoted by different operations in the same period t.

4.3.1. CPU load-based covert channel

In CPU load-based CC, two VMs start up as the sender and the receiver. A web server *Apache 2.2.6* is running in the receiving VM, where a single 48K-byte text-only HTML page is made publicly accessible. On the other side, *JMeter* is installed in the sending VM, which is a pure Java desktop application designed to load test functional behavior and measure performance. Then, the sender controls the CPU loads of the receiving VM, and the transmission cycle is described as follows.

- First, when bit 1 is transmitting, the sender in sending VM starts 100 threads simulating 100 single users and each thread takes 10 load samples. This burst HTTP requests take about $t = 10$ s, and the CPU load is obviously higher than 50%.
- Then, when bit 0 is transmitting, the sender in sending VM pause for $t = 10$ s. The CPU load is obviously lower than 50% during this period.
- The sender starts HTTP requests and pauses for t seconds according to the transferred binary string, and each request is made as fast as possible. This transmission cycle is repeated until all the bits have been sent out.

An example of the CPU load-based CC without noise is shown in Figure 7. The difference is obvious between the load samples when performing HTTP requests and not. The burst HTTP requests consume almost 70% of the CPU load and almost 0% on the contrary. The transmission continues for 3200 s, where each $t = 10$ s is denoting a bit. The receiver can easily decode the confidential information from the changes of CPU loads.

4.3.2. Cache-based covert channel

The cache-based CC sender and receiver are located in the different VMs but the same hypervisor. Two L2 caches are in the test bed, each serving two CPU cores. Each L2 cache is 12-way set-associative with $m = 3072$ cache sets and a line size of $l = 64B$, yielding a cache size of $c = 3$ MB. Each of the sending and receiving VMs is allocated a single vCPU, and the Prime–Probe protocol is implemented as follows.

- Firstly, the receiver reads a 3072-byte file to fill the entire cache and investigates the access time.

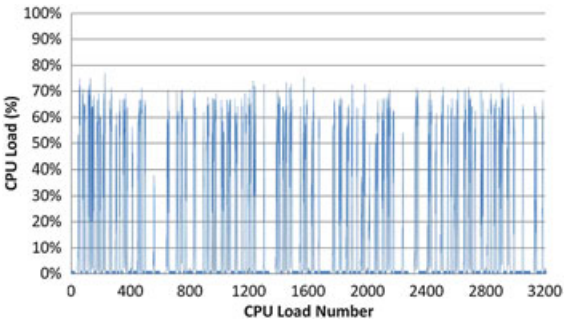

Figure 7. CPU load of covert channel without noises.

- At the same time, the sender starts tasks to occupy and release the cache in each $t = 1$ s period according to the transferred binary bits.
- If the cache is occupied, the receiver will take about 1 s to read the file. Otherwise, if the cache is free to use, the receiver will take about 0.5 s to read the file.
- This transmission cycle is repeated until all the bits have been sent out.

An example of the cache-based CC without noise is shown in Figure 8. The cache access time is obviously different from which the receiver decodes the confidential information.

4.3.3. Shared memory-based covert channel

In shared memory-based CC, the receiver and the sender are located in the different VMs but sharing the same memory. The sender and the receiver have predetermined the size of the *buffer_list* and cooperate in the producer and consumer model. The sender writes data into the memory buffer ring in certain intervals according to the binary bits. The explicit communication protocol is described as follows.

- The receiver set ups a shared ring structure with a grant table and passes the grant reference to the sender.
- The sender computes the intervals and then sends data according to the intervals in each transmission cycle.
- The receiver obtains the data and computes the intervals on each interrupt.
- This transmission cycle is repeated until all the bits have been sent out.

An example of SMCTC without noise is shown in Figure 9. The memory writing intervals are from almost 2000 to 10 000 μs. When bit 0 (or 1) is transmitted, the interval is short (or long). When all the intervals have been obtained by the receiver, the confidential information can be decoded.

4.4. Detection analysis

C²Detector is tested by the three types of CCs. Taking shared memory-based CC for example, a test sample includes 100 SMCTC and 100 normal operation sequences. Each sample is repeated 100 times to get the average detection performance. When the experiment is executing, some operations are running in the third VM to bring some noise, which causes the receiver to decode the message with some errors.

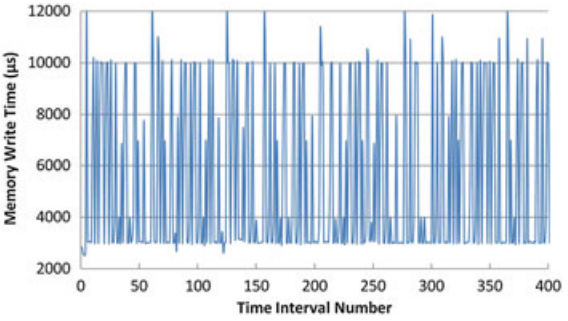

Figure 9. Time intervals of SMCTC without noises.

In SMCTC, the sender process P_i runs in the user space and then calls the shared ring using hypercall in Xen. P_i writes data into the shared ring and the receiver process P_j reads data from it. The call branches are expressed as follows, where $\odot$ means the action of operating, $\rightarrow$ means writing action, and $\leftarrow$ means reading action.

$$P_i \odot \{Share\ \ Ring\} = \{user\_write \rightarrow send\_request \rightarrow RING\_PUSH\_REQUESTS\_AND\_CHECK\_NOTIFY \rightarrow Shared\ \ Ring\},$$
$$P_j \odot \{Share\ \ Ring\} = \{user\_read \leftarrow get\_request \leftarrow RING\_GET\_REQUEST \leftarrow Shared\ \ Ring\}$$

The aforementioned call branches are captured as the operational records and flow into the Markov detector. The results of the Markov detector are shown in Figure 10. The value domain of $P(O|\lambda)$ is

$$0.078955 \leq P(O|\lambda) \leq 0.228742$$

As described in Section 3.5, the CCs have a small value of $P(O|\lambda)$. To lower false negative rate, a pessimistic threshold is set as

$$P_{\text{thr}} = 0.228742$$

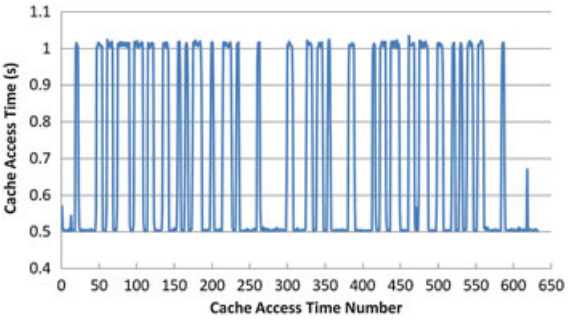

Figure 8. Cache access time of covert channel without noises.

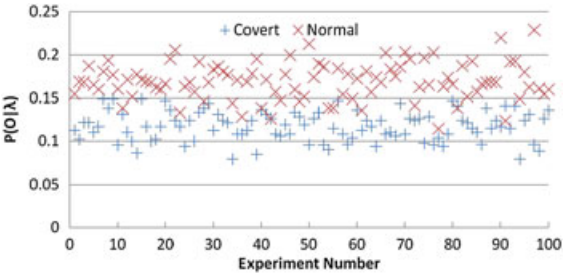

Figure 10. Markov detector results in detecting SMCTC.

Therefore, all the CCs satisfying the following condition

$$P(O|\lambda) < P_{\text{thr}}$$

are potential CCs. Because of P_{thr} being pessimistic, some normal operations sequences are included in this result by mistaken.

The sample has been repeated 100 times as shown in Figure 11. The false positive of the detectors in detecting SMCTC are shown in this figure. The false positive of the Markov detector is 16%, meaning 16 normal sequences have been detected by mistaken. Comparing with the Markov detector, the false positive of Bayesian detector is 10% in detecting the same sample. It is believed that the performance of Bayesian detector is better than the Markov detector when used independently.

The detector detects the sample use of Markov detector and Bayesian detector sequentially. Therefore, the false positive falls to 1%. The reason for the reduction is that the Bayesian detector refines the resulting output by the Markov detector. Only the channels that have been taken as covert by both the detectors are output as CCs.

The experiments of CPU load-based and cache-based covert channels are detected in the same way. Table I shows the average detection results of all the three types of CCs.

In Table I, capacity means the threat performance of a CC. The bigger capacity means that the CC is more threatening to the cloud computing. It can be seen that the shared memory-based CC is the most threatening to cloud computing. Error means the decoding error. In the experiments, it is calculated by edit distance [48], which is the minimum distance between two strings. The values of Markov and Bayesian are the false positive of each detector. In these experiments, the Bayesian detector is always better than the Markov detector. When the two detectors detect the channels sequentially, the final false positive is acceptable.

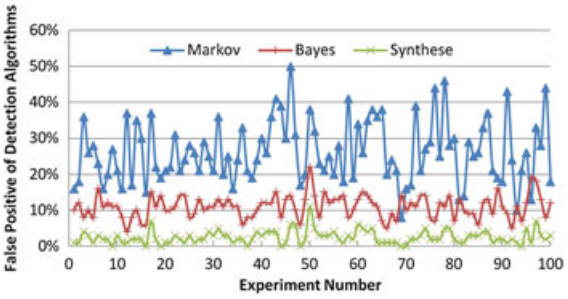

Figure 11. False positives of the Markov detector in detecting SMCTC.

These experiments show that the two-phase synthesis algorithm is efficient to detect CCs. C^{2}Detector framework detects all the three types of channels in cloud computing with no false negative and low false positive.

5. DISCUSSION

The CC is modeled into an error-corrected four-state automaton, adopting the probability-based error-corrected algorithms referred in Section 3.4 to process the errors in the experiments. When an error occurs, it takes the error as bit 1 (or 0) depending on the probability of the property states in Markov detector and Bayesian detector. Although this error-corrected algorithm simplifies the detection process, the detection accuracy may be affected. A more sophisticated algorithm will be developed in the future.

It has been shown that all the three types of CCs can be detected by C^{2}Detector framework. XML file is used as a plug-in to configure the detector to detect CCs. A typical XML file includes the channel name, the property domain values, and the pessimistic threshold. An example is shown in Figure 12.

This information is collected and appended to the XML file, so it is convinced that C^{2}Detector can detect the incoming channels as long as they can be modeled into this model. More channels will be built to test whether C^{2}Detector can detect or not in the future work.

To protect the privacy of the cloud customers, cloud providers should not collect the users' data. In C^{2}Detector, only the change regularities of the shared resources are concentrated, for example, CPU load, cache access time, and the memory writing intervals. It is no need to know which operation triggers these changes. The ignorance of the channel details protects the users' privacy and lowers the performance influence to hypervisor.

Only the channels using the binary encoding scheme are investigated and C^{2}Detector is easily extended to support multi-encoding channel [21]. A multi-encoding channel will be modeled into error-corrected multi-state automata in the same way. Each codeword is modeled into a state in the automata and can be transited from all the other states. The corresponding Markov detector can be modified easily. There is no difference between multi-encoding and binary channels, and C^{2}Detector detects this type of channels in the same way as the binary ones.

Table I. Detection results of the three channels.

Channels	Capacity (bps)	Error rate (%)	Markov (%)	Bayesian (%)	False positive (%)
CPU load based	0.098	4.76	29.92	8.14	4.46
Cache based	0.297	6.67	24.51	15.61	1.89
Shared memory based	189.15	2.43	26.28	16.42	2.53

```
<covert channel>
   <channel name = "CPU load based covert channel">
      <dom>
        <sender id = "..."/>
        <receiver id = "..."/>
      </dom>
      <Markov>
        <pessimistic threshold="..."/>
      </Markov>
        ......
    </channel>
    ......
</covert channel>
```

Figure 12. XML configuration File in C²Detector.

C²Detector is a plug-in framework; whenever a new efficient detection algorithm is discovered, it can replace the existing detector or be added as another detector. However, C²Detector is built to detect CCs only, and other intrusion methods cannot be detected. C²Detector may be integrated into other intrusion detection tools in the future.

6. RELATED WORK

Most of the prior works on cloud computing security have focused on the integrity protection. Several frameworks have been proposed to protect the integrity of the guest kernels using hypervisor. For example, HIMA [49], Lares [12], HyperSentry [13], HyperSafe [9], and Antfarm [50] are all designed to provide the integrity protection to VMs. These approaches are related to the virtualization security and integrity, but only C²Detector focuses on the CCs.

Millen presents a CC detection approach, but it is limited to the covert storage channels [37]. This paper aims to detect timing channels in cloud computing, which is more difficult. Cabuk and Berk present the detection algorithms to detect network covert timing channels [26–28]. The CCs they detected are classified into CC2 type of channels, which has been studied in depth for many years [20,21,51–53]. In this paper, only CC3 type of channels that are recently brought by cloud computing are considered.

Nagatou *et al.* [29] present a run-time detection approach and focus on enforced properties. C²Detector can also perform online detection. HomeAlone [35] is proposed to determine whether a VM is physically co-resident to another VM, and the same functions are proposed in [7,34]. Although these approaches are related to CCs, they are not intended for detecting them.

Some introspections used as intrusion detection systems have been proposed upon VMs. For example, HyperSpector [54] is designed to monitor the actions without any additional hardware by using virtualization to isolate each Instruction Detection System (IDS) from the servers. Some other IDSs are presented in [55–57]. CC can be seen as a type of intrusion, and C²Detector is a special detector only detecting the CCs. C²Detector is easy to extend and has better detection performance, which is a complement to these introspections.

7. CONCLUSION

With the growing popularity of cloud computing, more confidential applications have been deployed in the cloud. CC is a serious threat to the data of cloud customers. In this paper, the fundamentals to the CCs in the cloud are argued, and the channels are classified into three categories. Only the new channels brought by cloud computing are concerned and modeled into an error-corrected four-state automaton. Some formal requirements are presented to build a CC detector and a plug-in detection framework named C²Detector is designed.

C²Detector satisfies the four formal requirements proposed in Section 3.2 by using two key techniques. The first technique installs hooks inside the hypervisor to monitor all the hypercalls and capture the operation tracks. The second key technique is the two-phase synthesis detection algorithm implemented as Markov detector and Bayesian detector. To evaluate C²Detector, a prototype is implemented on Xen hypervisor. The CPU load-based, cache-based, and shared memory-based CCs have been implemented and detected by C²Detector. The results show that C²Detector can detect all the three types of the channels using the pessimistic threshold and has a small false positive rate. In addition, this research demonstrates that C²Detector is highly efficient and feasible to extend to detect the incoming new channels.

Cloud security is a hot topic, and CCs are a new security to cloud computing. Therefore, there are several interesting future directions for this work, as improving C²Detector to detect other types of channels, studying on the other detection algorithms to lower the false positive rate, combining C²Detector with intrusion detection systems, and all of these will be investigated continually.

ACKNOWLEDGEMENTS

This work is supported by the National Science and Technology Major Project No. 2012ZX01039-004 and No. 2010ZX01036-001-002, the National Natural Science Foundation of China No. 61170072, and the Major Program of the National Natural Science Foundation of China No. 91124014. Samee U. Khan's work was partly supported by the Young International Scientist Fellowship of the Chinese Academy of Sciences, (Grant No. 2011Y2GA01).

REFERENCES

1. Vaquero LM, Rodero-Merino L, Caceres J, Lindner M. A break in the clouds: towards a cloud definition. *SIGCOMM—Computer Communication Review* 2008; **39**:50–55.
2. Armbrust M, Fox A, Griffith R, *et al*. A view of cloud computing. *Communications of the ACM* 2010; **53**(4): 50–58.
3. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 2009; **25**:599–616.
4. Barham P, Dragovic B, Fraser K, *et al*. Xen and the art of virtualization. In *SOSP*, 2003; 164–177.
5. Takabi H, Joshi JBD, Ahn G-J. Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy* 2010; **8**(6):24–31.
6. Chen Y, Paxson V, Katz RH. What' s new about cloud computing security? EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-5, 2010.
7. Ristenpart T, Tromer E, Shacham H, Savage S. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *ACM Conference on Computer and Communications Security*, 2009; 199–212.
8. Wu J, Ding L, Wang Y. Research on key problems of covert channel in cloud computing. *Journal of Communications* 2011; **32**(9A):184–203.
9. Wang Z, Jiang X. Hypersafe: a lightweight approach to provide lifetime hypervisor control-flow integrity. In *IEEE Symposium on Security and Privacy*, 2010; 380–395.
10. Payne BD, Sailer R, Cáceres R, Perez R, Lee W. A layered approach to simplified access control in virtualized systems. *Operating Systems Review* 2007; **41**(4):12–19.
11. Sailer R, Jaeger T, Valdez E, *et al*. Building a MAC-based security architecture for the Xen open-source hypervisor. In *ACSAC*, 2005; 276–285.
12. Payne BD, Carbone M, Sharif MI, Lee W. Lares: an architecture for secure active monitoring using virtualization. In *IEEE Symposium on Security and Privacy*, 2008; 233–247.
13. Azab AM, Ning P, Wang Z, Jiang X, Zhang X, Skalsky NC. Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In *ACM Conference on Computer and Communications Security*, 2010; 38–49.
14. Aviram A, Hu S, Ford B, Gummadi R. Determinating timing channels in compute clouds. In *CCSW '10: Proceedings of the 2010 ACM workshop on Cloud Computing Security Workshop*. New York, NY, USA: ACM, 2010; 103–108.
15. NCSC. Trusted computer system evaluation criteria (orange book), 1985.
16. Lampson BW. A note on the confinement problem. *Communications of the ACM* 1973; **16**(10):613–615.
17. Wu J, Ding L, Wang Y, Han W. A practical covert channel identification approach in source code based on directed information flow graph. In *IEEE SSIRI*, Jeju Island, Korea, 2011; 98–107.
18. Tsai C-R, Gligor VD, Chandersekaran CS. A formal method for the identification of covert storage channels in source code. In *IEEE Symposium on Security and Privacy*, 1987; 74–87.
19. Keefe TF, Tsai W-T, Srivastava J. Database concurrency control in multilevel secure database management systems. *IEEE Transactions on Knowledge and Data Engineering* 1993; **5**(6):1039–1055.
20. Zander S, Armitage GJ, Branch P. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys and Tutorials* 2007; **9**(1–4):44–57.
21. Wu J, Wang Y, Ding L, Liao X. Improving performance of network covert timing channel through Huffman coding. *Mathematical and Computer Modelling* 2012; **55**(1–2):69–79.
22. ISO/IEC. Common criteria for information technology security evaluation, 2005.
23. Wang Y, Wu J, Zeng H, Ding L, Liao X. Covert channel research. *Journal of Software* 2010; **21**(9):2262–2288.
24. Wu J, Wang Y, Ding L, Zhang Y. Constructing scenario of event-flag covert channel in secure operating system. In *ICIMT*. Hongkong, 2010; 371–375.
25. Tsai C-R, Gligor VD. A bandwidth computation model for covert storage channels and its applications. In *IEEE conference on Security and privacy*. Oakland, California, 1988; 108–121.
26. Cabuk S, Brodley CE, Shields C. IP covert timing channels: design and detection. In *ACM Conference on Computer and Communications Security*, 2004; 178–187.
27. Cabuk S, Brodley CE, Shields C. IP covert channel detection. *ACM Transactions on Information and System Security* 2009; **12**(4):1–29.

28. Berk V, Giani A, Cybenko G, Hanover N. Detection of covert channel encoding in network packet delays. *Rapport technique TR536, de l'Université de Dartmouth. November*, 2005.
29. Nagatou N, Watanabe T. Run-time detection of covert channels. In *ARES*, 2006; 577–584.
30. Hélouët L, Roumy A. Covert channel detection using information theory. In *SecCo*, 2010; 34–51.
31. Girling CG. Covert channels in LAN's. *IEEE Transactions on Software Engineering* 1987; **13**(2):292–296.
32. Wu J, Ding L, Wang Y, Han W. Identification and evaluation of sharing memory covert timing channel in Xen virtual machines. In *IEEE CLOUD*. IEEE Computer Society: Washington DC, USA, 2011; 283–291.
33. Wu J, Ding L, Lin Y, Min-Allah N, Wang Y. Xenpump: a new method to mitigate timing channel in cloud computing. In *IEEE CLOUD*. IEEE Computer Society: Hawaii, USA, 2012; 678–685.
34. Okamura K, Oyama Y. Load-based covert channels between Xen virtual machines. In *SAC*, 2010; 173–180.
35. Yinqian Zhang AO, Juels A, Reiter MK. Homealone: co-residency detection in the cloud via side-channel analysis. In *IEEE Symposium on Security and Privacy*. Oakland, California, 2011; 313–328.
36. Chisnall D. *The Definitive Guide to the Xen Hypervisor*. Prentice Hall Press: Upper Saddle River, NJ, USA, 2007.
37. Millen JK. Finite-state noiseless covert channels. In *CSFW*, 1989; 81–86.
38. Lanotte R, Maggiolo-Schettini A, Troina A. Time and probability-based information flow analysis. *Software Engineering, IEEE Transactions on* 2010; **36**(5):719–734.
39. Rabiner LR. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 1989; **77**(2):257–286.
40. Hu J, Yu X, Qiu D, Chen H-H. A simple and efficient hidden Markov model scheme for host-based anomaly intrusion detection. *IEEE Network* 2009; **23**(1): 42–47.
41. Xie Y, Yu S. Monitoring the application-layer DDoS attacks for popular websites. *IEEE/ACM Transactions on Networking* 2009; **17**(1):15–25.
42. Mitchell TM. *Machine Learning*. McGraw-Hill: New York, NY, USA 1997.
43. Moore AW, Zuev D. Internet traffic classification using Bayesian analysis techniques. In *SIGMETRICS*, 2005; 50–60.
44. Auld T, Moore AW, Gull SF. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks* 2007; **18**(1): 223–239.
45. Son J, Alves-Foss J. A formal framework for real-time information flow analysis. *Computer Security* 2009; **28**(6): 421–432.
46. Osvik DA, Shamir A, Tromer E. Cache attacks and countermeasures: the case of AES. In *CT-RSA*, 2006; 1–20.
47. Tromer E, Osvik DA, Shamir A. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology* 2010; **23**(1):37–71.
48. Ristad ES, Yianilos PN. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1998; **20**(5):522–532.
49. Azab AM, Ning P, Sezer EC, Zhang X. HIMA: a hypervisor-based integrity measurement agent. In *ACSAC*, 2009; 461–470.
50. Jones ST, Arpaci-Dusseau AC, Arpaci-Dusseau RH. Antfarm: tracking processes in a virtual machine environment. In *USENIX Annual Technical Conference, General Track*, 2006; 1–14.
51. Yao L, Zi X, Pan L, Li J. A study of on/off timing channel based on packet delay distribution. *Computers & Security* 2009; **28**(8):785–794.
52. Chen S, Wang R, Wang X, Zhang K. Side-channel leaks in web applications: a reality today, a challenge tomorrow. In *IEEE Symposium on Security and Privacy*, 2010; 191–206.
53. Wang Y, Wu J, Ding L, Zeng H. Detection approach for covert channel based concurrency conflict interval time. *Journal of Computer Research and Development* 2011; **48**(8):1542–1553.
54. Kourai K, Chiba S. Hyperspector: virtual distributed monitoring environments for secure intrusion detection. In *VEE*, 2005; 197–207.
55. Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection. In *NDSS*, 2003.
56. Jiang X, Wang X. "Out-of-the-box" monitoring of VM-based high-interaction honeypots. In *RAID*, 2007; 198–218.
57. Li J, Li B, Wo T, *et al*. Cyberguarder: a virtualization security assurance architecture for green cloud computing. *Future Generation Computer Systems* 2012; **28**(2):379–390.