11-2016

A novel covert channel detection method in cloud based on XSRM and improved event association algorithm

Lina WANG
Wuhan University

Weijie LIU
Wuhan University

Neeraj KUMAR
Thapar University

Debiao HE
Wuhan University

Cheng TAN
Wuhan University

See next page for additional authors

A novel covert channel detection method in cloud based on XSRM and improved event association algorithm

Lina Wang[1,2], Weijie Liu[1,3], Neeraj Kumar[4], Debiao He[1,2], Cheng Tan[1], and Debin Gao[3]

[1]Computer School, Wuhan University, Wuhan, China
[2]Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education, Wuhan, China
[3]School of Information Systems, Singapore Management University, Singapore
[4]Department of Computer Science and Engineering, Thapar University, Patiala, India

Abstract

Covert channel is a major threat to the information system security and commonly found in operating systems, especially in cloud computing environment. Owing to the characteristics in cloud computing environment such as resources sharing and logic boundaries, covert channels become more varied and difficult to find. Focusing on those problems, this paper presents a universal method for detecting covert channel automatically. To achieve a global detection, we leveraged a VM event record mechanism in Hypervisor to gather necessary metadata. Combining the shared resources matrix methodology with events association mechanism, we proposed a distinctive algorithm which can accurately locate and analyze malicious covert channels from the respect of behaviors. Compared to the popular statistical test methods focusing on the single covert channel, our method is capable of recognizing and detecting more covert channels in real time. Experimental results show that this method is not only able to detect multi-level and multiform covert channels in cloud environment effectively, but also facilitates the implementation and deployment in practical scenarios without modifying the existing system.

Keywords: Cloud Security; Covert Channel Detection; Shared Resource Matrix; Event Association Analysis

1 Introduction

Cloud Computing has raised great concern in industry and academia for its low cost, rapid deployment, flexibility and other advantages. The characteristics such as resource dynamic aggregation, autonomous collaboration and services outsourcing has turned the cloud boundary into an ambiguous, generic and dynamic boundary, along with new technical challenges [1]. Traditional information systems infrastructure is relatively static: there are obvious physical boundaries between system inside and outside. By deploying access control, intrusion detection, firewall, security audit and other mechanisms, we can effectively protect the security of conventional IT infrastructures. However, in a public cloud environment, physical separation between different data owners' networks are replaced by virtual separation (e.g., in IaaS) and security threats are still rampant.

As one of the core technologies of cloud computing, virtualization security has attracted a lot of attention naturally. Under normal circumstances, malicious programs are difficult to disclose data between virtual machines due to the isolation provided by hypervisor [2]. Unfortunately, the attacker can use a variety of covert channel to bypass the security mechanisms for information leakage [3]. Okamura discovered a CPU-load based covert channel in cloud, which not only enabled exploitation of CPU load to transfer private data stream, but also managed to avoid being detected [4]. Salaun mentioned that covert channels might exist on Xen, from XenStore sharing page agreement, driver loading, and event data transmission [5]. Since Ristenpart [6] and Zhang [7] successfully exploited side channel attack to obtain the target guest virtual machine's private information in Amazon EC2, side channel exploitation based on virtual machine co-residency becomes a hot topic. Researchers gave their solutions [8, 9, 10, 11]. And these studies take the same view that manual maintenance and monitoring has met more difficulties in cloud. The demand for automatic detection of covert channels also becomes more urgent.

Massive parallelism in the cloud makes covert channels pervasive and hard to detect. On the other hand, the implementation of the hardware features is usually architecture-specific and has changed amazingly fast over various versions of CPU and cache. Any exclusive channel detection on shared hardware resources will be old-fashioned. Cloud platforms need an automated, extensible, multi-faceted and comprehensive tool to better coping with new attacks caused by malicious covert channel exploitation. However, our solution is dedicated to solving those problems.

We proceed as follows. Section 2 discusses the related works. Section 3 introduces the features of covert channel exploitations in cloud and gives the threat assumption. Section 4 sketches our design and how these features can be used to detect a covert channel. Detailed implementations for the design are explored in Section 5. In order to evaluate the correctness and effectiveness of our approach, we presented our experimental results from a simulated prototype of our solution in Section 6, and Section 7 concludes.

2 Related work

The initial concept of covert channel was proposed by Lampson in 1973 [12]. Follow-up studies divided covert channels into two categories: storage channel and timing channel [13]. Focusing on the reason why covert channel forms, a series of covert channel identification methods were proposed, including information flow analysis [14], non-interference model [15], shared resources matrix method [16] and code-level analysis techniques [17].

However, in cloud computing environment, covert channel recognition and detection becomes a more difficult issue. It is extremely complex to analyze the increasing code lines in virtualization systems, while it is lack of effective automated tools [18]. Secondly, the traditional covert channel detection methods are not suitable for cloud computing environment, which need for top-level design analysis or source code analysis. Moreover, traditional methods cannot guarantee real-time processing. Thirdly, the exploitation of covert channel is facing a lot of interference such as extremely short time interval and overlapped noise.

Bates et, al. proposed co-residency watermarking as a covert channel in virtualization environment and put forward corresponding detection approach [19]. Varadarajan et al. investigated the problem of placement vulnerabilities and quantitatively evaluate three popular public clouds for their susceptibility to co-

location attacks [20]. At the same time, they found ways to detect co-location with victim web servers located behind a load balancer. Wu et al. proposed C2Detection [21], which captures information flow in hypervisor layer and combines Markov and Bayesian model to detect hidden channels.

Chen et al. present a mechanism called time deterministic replay (TDR) [22] that can reproduce the execution of a program, including its precise timing [23]. Without TDR, reproducing the timing of an execution is difficult because there are many reasons to cause timing variation, such as preemptions, hardware interrupts, cache effects, scheduling decisions, etc. Our method uses VM record mechanism as well, but we record hardware events for further analysis rather than for replay, so that we can omit a number of unnecessary operations which on the contrary are vital in re-execution of a VM. For example, in order to ensure the accuracy of the execution result, the replay system needs to distinguish different VM's events, i.e. when each VM Exit/VM Entry happens, the system must reset the performance counter and re-count every VM's branch counter, which will make a plenty of overheads.

From what has mentioned above, most recent schemes start from a single covert channel, characterizing the feature on the face of it, whose accuracy is unsatisfactory. In cloud, following problems will appear when detecting covert channels:

Firstly, in addition to the traditional ones, there are drastically increasing amounts of special channels which cannot be recognized easily in cloud environment. Secondly, covert channel analysis becomes more difficult in supersize cloud platform with growing code lines. Thirdly, traditional covert channel analysis method desperately relies on manual work. While taking the SLA (Service-Level Agreement) into account, manual analysis cannot guarantee the service continuity, convenient implementation, and high efficiency.

3 Covert channels in cloud

As we know, it was possible to control timing channels by limiting untrusted processes' access to high-resolution clocks in the days of uni-processor and single-threaded process [24]. Nevertheless, in cloud environment, inside attackers become outsider attackers, and data leakage is unlikely to be caught. Likewise, it is not easy to accomplish mitigation on the basis of malware detection. We should find some other ways to identify those malicious channels.

3.1 Clocks and events

A broad definition of clock is that any approach for measuring the progress of time can be referred to as a clock [25]. One pair of clocks is necessary to data transmission, since one clock should be a measure clock, the other one should be used to issue signals, or we say, to perform the occurrence of events. We view the passage of time as being characterized by sequences of events which can be distinguished one from another by an observer.

There are four possible clock sources in computer systems [26]: the CPU instruction-cycle clock, the real-time clock, the I/O subsystem (completion interrupts, DMA data arrival rate, etc.) and the memory subsystem (data/instruction fetch, interlocks, etc.). Most of these clocks could be modulated. For example, the memory subsystem provides a clock based on the time take to perform a memory fetch. This time depends on whether the target data was in the cache (and, for a system with a multi-level cache, which cache it was in), and on the level of traffic on the memory bus and through the memory controller. Some of these clocks are not independent. Event sources often communicate with the external world by

accepting input. Especially the massively parallel context in cloud creates numerous implicit high resolution clocks. As an example, consider a process using DMA data arrival to construct a clock. The process issues a disk read request into a buffer, and then polls the first byte of the buffer. When the data in that byte changes, the process knows the DMA transfer has started. It then polls another location, further along in the buffer, and when the data changes, the process knows that the transfer has reached the other point in the buffer.

Generally speaking, the stream of events can be considered as a clock, and these events appear at anywhere and anytime, such as during memory accesses, with I/O operations, and in DMA channels.

3.2 Events and covert channels

As we know, a peer to peer communication consists of three essential factors, $\langle sender, receiver, channel \rangle$. So does a covert channel exploitation. However, a covert channel has some additional special properties. Firstly, the sender has secret modulation actions. Secondly, the receiver can perceive issuance of the sender's signals. Thirdly, both parties of communication use a variable or multiple variables to transmit information. Whether a given event can be distinguished from another depends on the observer. For instance, if an event consists of a boolean changing variable, then there are only two intrinsically distinguished events. However, if an observer is incrementing a counter of how many times the variable has changed state, then there are an infinite number of possible distinguished events for that observer. The presence of sufficient internal memory allows an observer to distinguish arbitrary numbers of otherwise indistinguishable events.

From the relationship between clocks, events and covert channels, we can see the ability of events as a clock in transmitting secret information. Then we can envision that event association analysis will be an effective solution of covert channel detection in nature. So our scheme takes the advantage of event association analysis to discover covert channel exploitations (described in Section 4).

3.3 The perspective of observers

Since it has been explained that the event can be perceived and modulated in information system, then it is possible to build a covert channel by using some specific events. However, how do we determine whether an event can be used as an information carrier in channel exploitation? It involves observation perspectives.

Perspectives in observation fall into many categories. We regard the perspective of the sender (i.e. the modulator of signals) as interior where we can control the internal timing, and regard the perspective of the receiver (i.e. the demodulator of signals) as exterior [27]. The choice of observation perspective is significant. When our perspective varies, the exploitability of the corresponding channels may be different, and the bandwidth and the noise may vary greatly as well [28].

For example, there is an ordinary and common bus contention-based channel on most virtualization platforms. To understand the procedures of this covert channel exploitation, consider the following scenarios. We assume that the sender and the receiver have been able to control the memory bus, and their operations are strictly alternate. When we inspect the CPU load, we can perceive intensive bus contentions. That is because when the sender occupies the memory bus frequently, the CPU load will be much higher than that in a normal situation. However, this observation approach is not accurate, and the bandwidth of this kind of covert channel exploitation is low. Another way of observation is cancelling the

output. When using this approach, the receiver requests a memory bus related task, and cancels the task after a certain time, then observes the output. Because the task execution and other references to bus are asynchronous, if there is a contention bus, then the cancellation will be postponed. Such a delay can be perceived by the external receiver because there is a transition phase in the memory, and it is possible to be observed or perceived by other requests. The information obtained in this way of observation is completely different from the previous one. This covert channel exploitation has a better grasp of time and a higher bandwidth.

After all, in terms of covert channel detection, we should take a global point to discover unknown channels at behavior level to the greatest extent. Therefore, in our solution, we utilized a global observation scheme which is deployed on the Hypervisor layer by presenting machine environment and operations in VM events.

3.4 Threat assumption in cloud

Owing to physical isolation boundaries between traditional physical hosts, we have to communicate through networks in traditional computing environment. However, in virtualization environment/cloud computing environment, there are logical boundaries between virtual machines that belong to the identical physical host. VMs can exchange data in many other ways (in Figure 1). Malicious virtual machines can leverage these covert channels to steal sensitive information out of other guest virtual machines.

In [21], three kinds of covert channels were discussed, which are intra-domain, inter-domain (cross-VM) and cross-platform covert channels. For intra-domain covert channels, the hidden process can be detected via Hypervisor [29, 30]. For cross-platform covert channels, processes P_i and P_j can only communicate by network. This kind of covert channels can be detected with the method based on Markov model or others [31]. So our work just aims at inter-domain (cross-VM) channels and does not mention any deeper researches about those intra-domain and cross-platform channels. Here we give the further categorization:

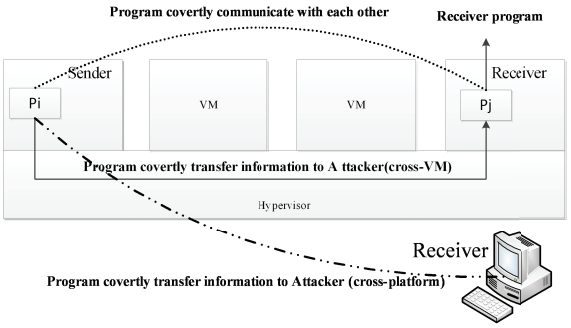

Figure 1: Collusive attack model in cloud

(i) Hardware layer covert channels for physical layer attacks.

Normally, this kind of covert channel exploitations take advantage of the timing signal caused by CPU scheduler, memory or cache preemption, and I/O interrupts [4, 32], or leverage the access latency caused by memory missing page or memory bus contention [33] to transfer information. Moreover, they may construct covert channels based on cold boot attacks or just some parts of shared cache and unprotected memory.

(ii) System-level covert channels for Hypercall and VMM sharing mechanism attacks [34].

To accomplish the inter-domain communication and cooperation, the virtualization platform generally uses sharing resources methods, and that the most common ways are event channel, Foreign Map mechanism [35] and cross-VM sharing resources [36]. If the sender domain controls the fill-

ing timing of sharing memory and the receiver domain observes the time when obtains data, then it can construct timing channels according to the features of time uncertainty.

As a result of the above categorization, we give covert channel definitions in cloud environment:

Definition 1 Covert Channel

$\langle V,\ T,\ A,\ Ph,\ Pl \rangle$

$V = \{variable \mid$ Each variable represents one kind of resources in current cloud computing system.$\}$

$T = \{type \mid$ Each type represents one category of covert channels in cloud environment.$\}$

$A = \{attribute \mid$ Each attribute represents one special feature of a covert channel.$\}$

Ph and Pl, are denoted as the two parties which have different security levels in covert channel communication. Differed from in traditional computing environment, there are several levels in cloud. Therefore, many available perspectives can be adopted in observation, resulting in more variables. When you want to detect covert channel exploitations, it may be difficult to locate them accurately. Besides, because all covert channels are constructed by the knowledge of system asynchronous events, so the storage nature and timing nature can be the attributes of a covert channel. If we introduce the channel type T and attribute A, we can divide the covert channels into various categories by using the predefined types and attributes. This can greatly improve the efficiency and correctness of our detection solution.

T and A can be defined by expert system or cloud providers; *type* refers to the category of a covert channel. Namely it represents whether it is a timing channel or storage channel, a memory leak channel or an inter-VM channel, etc. Then, *attribute* refers to some obscure indicators and properties of a covert channel. For example, attributes in A usually consist of the characteristics of a covert channel in communications, such as the conditional entropy, throughput, etc. And if the channel is constructed by shared hardware resources, then A may contain CPU load, I/O interrupt interval, the timing of VM Exit/VM Entry, etc.

4 A novel covert channel detection method in cloud computing environment

According to the above idea, we proposed a new covert channel detection method. The distinguished contributions made in this paper are as follows:

(i) The method is different from traditional ones which analysis time series such as access interval time by using statistical models like Markov model or Bayes classifier [21]. We take a new way, from the perspective of behavior, which takes the fundamental characteristics of covert channels into account to detect covert channel exploitations.

(ii) Due to the traditional SRM method is unrealistic to analyze covert channel [25], we improved the method so that it can be used in real time covert channel detection. Furthermore, we proposed the event correlation analysis method based on XSRM (extended shared resources matrix).

On the basis of the matrix data structure, we made some improvements for our event correlation analysis algorithm, which makes it more suitable for the practical application. That is because: firstly, matrix data structure can be transformed to vertical data structure that is more suitable for the association mining algorithm;

secondly, we optimize the algorithm by using the idea of partitioning, greatly reducing system overhead.

(iii) In our method, we leveraged VM record mechanism [22] and VMI (Virtual Machine Introspection) [37] to collect VM's behaviors. Underlying events will be logged to make further analysis. On the other hand, the method is applied on the VMM layer, which means it is transparent for guest-VMs. Our method also has a global view over all external observers and thus has a better understanding of guest-VM's behaviors. It is beneficial for discovering most cross-VM covert channels.

4.1 Framework overview

In order to describe such method in a better way, we present our detection framework, as shown in Figure 2. Firstly, it collects information from cloud platforms, including VMM event logs, logs of some daemon processes, network configurations and security policies. Then, this framework extracts information in terms of a defined event format from the above-mentioned files. By using the shared resources matrix method and extracting the event metadata, the characteristics of covert channel exploitations could stand out. The third step is event merging. Associated events that comply with certain conditions will be merged into one event flow. In order to do that, we need to carry out association analysis for these events to find out the features inside, and establish a series of matching policies. And then, the framework matches the features of covert channels in the form of event flows. Finally, alarms are reported and the detection is accomplished.

4.2 Information gathering

By using active scanning mode, we collect multifarious files, which mainly include event logs and security configurations from cloud platforms. Based on that, we need to collect more detail data, including platform basic status (such as the pattern of contention on the hardware resources), task response delay, event logs, network configuration, network security strategies, network connections, traffic information and so on.

4.2.1 Event recording

To describe our framework more clearly, we defined events:

Definition 2 Event

Event $e = \langle eventId, time, vmId, processId, sharedVariableName, type, attributes, return \rangle$

Where, $sharedVariableName \in V$, which represents the object of the operation in an event; $type \in T$, which is used to describe which category of the event belongs; *attributes* refers to elaborative event features which have been defined in Definition 1. For hardware events, attributes may include $\langle CPU\ load, memory\ access\ interval \rangle$; For system events, attributes may include $\langle HypercallId, grantTableNumber \rangle$. For convenience, we define E as the collection set of all the events in cloud computing system. It is clear that the definition of events is extensible. We can add any fields we want to join in *attributes*.

Event recording plays a crucial role in our scheme. In virtualized platforms such as Xen and KVM, some logs of VMM can be obtained directly from daemon processes, for instance the logs of Xend and XenStore (detailed in Section 5.2). However, deterministic and non-deterministic events in some systems are not recorded. Therefore, we recorded these events by using VMI and VM record, and outputted them in the form of events for later analysis.

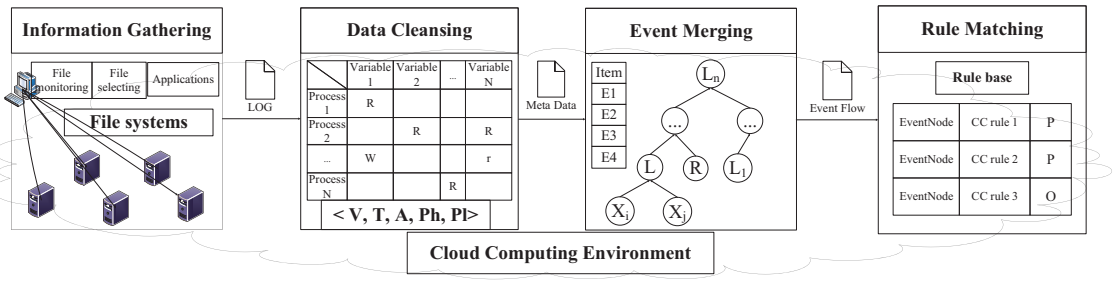

Figure 2: Framework Overview

4.3 Extended Shared Resources Matrix

After the previous information gathering and log file filtering steps, the condensed data was obtained. But the data is not only too massive and fragmented, but also useless for extracting attacker's behaviors. At this point, we used the Shared Resources Matrix method [16] for reference. Suspicious events can be extracted for further processing.

We extended the definition of shared resources matrix, and applied it to our scheme.

Definition 3 the eXtended Shared Resources Matrix

Assume S as the information system;

Define the extended shared resources matrix $XSRM = (a_{ij})_{m\times n}$. The XSRM is also a concept lattice having the following constrains:

In cloud environment, many data sets are mixed-data sets, which consist of both numerical attributes and categorical attributes. Here our XSRM is a mixed data table. More formally, XSRM is described by a quadruple $XSRM = (E, A, V, f)$ [38], where A and E have already defined in **Definition 1** and **Definition 2**. Meanwhile, A is a nonempty set of attributes with $A = A^r \cup A^c$. A^r is a numerical attribute set which includes CPU load, time interval of cache/memory access, and so on. A^c is a categorical attribute set which consists of shared variable's name, parameters of Syscall/Hypercalls, and so on; V is the union of attribute values, i.e., $V = \bigcup_{a\in A} V_a$, where V_a is the value domain of attribute a; $f : E \times A \to V$ is an information function such that, for any $a \in A$ and $e \in E$, $f(e, a) \in V_a$.

Fill in the events as shown in Table 1 (each event is filled in with the corresponding attributes as a line). Denote row $\{a_{i1}, a_{i2}, ..., a_{in}\}$ as $A(i, :)$, column $\{a_{1j}, a_{2j}, ..., a_{mj}\}$ as $A(:, j)$. Each row in XSRM represents an event in the current information system. Each column represents the states of an attribute at that timestamp. Meanwhile, there is an exception that the first column represents the $sharedVariable$, which is a special attribute and an extension of the concept lattice.

Table 1: eXtended Shared Resources Matrix

XSRM	*sharedVariable* a	attr b	attr c...
Event e1			
Event e2			
Event e3			

Usually, the shared resources matrix method is used to detect storage channels. However, it also can be used to detect timing channels after our improvement. That is because our scheme uses events instead of TCB primitives as column in original SRM method. Each column of XSRM will be operated in our event association analysis algorithm (detailed in Section 4.4.1). We make this modification mainly due to the following two points.

(i) Timing channels are directly related with asynchronous events. In other words,

there must be correlations between the sender process activities (i.e. asynchronous events) and the occurrence of two clocks. It is possible that their occurrence timings are close, or they used the same objects (shared variables), etc. Of course, this correlation exists not only in the procedure of attacker's signal modulation, but also in the procedure of demodulation.

(ii) Events, logged along with a program execution, provide the most intimate details of the program's code path. Whereas, in most virtualization systems, users of guest-VM cannot directly perceive external interrupts, but they can perceive some events that happened during interrupt handling processes. Our target is to capture the events that we can perceive inside, to compare the actual outside events, and to try our best to fill in this perception gap between the host and VMs, then we can determine the existence of covert channels.

4.4 Matching with association

Suspicious events acquired in the previous step are still mixed and disorderly, so they need to be merged into a single event sequence in order to better analyze the behavior extracted from the subject of events. And in that way, when a new sequence of events occurs, we can find the trace that covert channel left behind by matching the event flow with specific rules.

4.4.1 Event merging based on association analysis

Since the covert channel is also a kind of channels, it has the characteristics of a normal channel. Hence, causalities between attacker's behavior and resulting events exist when a covert channel forms. Namely, there are associations between the resulting events. As mentioned in Section 3, from the relationship between events and covert channel, we can understand that the event flow can be used as relative clocks in covert channel exploitations. And it may contain a lot of information about the channel. For instance, attackers who exploit the covert channel must perform synchronization and negotiation steps before the communication step, which will leave some obscure traces behind.

Besides, the sequence of events could represent the covert channel observer's (or receiver's) clocks. For that reason, our scheme tries to combine the features of the traces, which will be helpful to infer attackers' behaviors. These event sequence patterns are also helpful when building the matching rules [39]. In order to obtain those patterns, and based on **Definition 2**, we defined Associated Events and Event Flow:

Definition 4 Associated Events

$\forall Event\ e1, e2$

If $e1.type == e2.type$ && $e1.attributes[i] == e2.attributes[i]$ && $|e1.time - e2.time| \leqslant TW$

then $e1$ and $e2$ are considered associated, denoted as $e1 \Rightarrow e2$.

Definition 5 Event Flow

$EventFlow\ eventflow = \langle eventFlowName,\ eventFlowId,\ eventList,\ TW \rangle$

Wherein, $eventList$ is a linked list of event nodes, which represents the timing relationships between events in the corresponding event flow. $eventFlowType$ refers to a common type of all events in the event flow; TW (Time Window) indicates the maximum delay of the flow of events from the first event to the last.

Now it is necessary that we have to figure out the relationship between the events, association analysis algorithm is the best choice. However, note that the traditional data mining

algorithm does not work here, thus it needs to be improved in order to adapt to the application scenarios in cloud computing environment.

As a result, we proposed an improved frequent itemsets association analysis algorithm for XSRM based on vertical data format. The database for storing events is actually a Concept Lattice matrix, which can be joined with the shared resources matrix. The vertical data format can be processed efficiently in a frequent itemsets association algorithm, and such a matrix data structure is flexible to be transformed to vertical data format. In this paper, a combination scheme is adopted to solve the problem of a large data mining which commonly happens in real-world detection scenarios.

The improvements we have made are as follows.

(i) The introduction of shared resource matrix reduces the cost of the search for item merging to a minimum, which greatly improves the efficiency of the algorithm.

We searched frequent itemsets with vector "and" operation and generated event flows after merging the events. When the algorithm executes the intersection operation $Tidsets\,(R) = Tidsets\,(X_i) \cap Tidsets\,(X_j)$, because of this special data structure (the Concept Lattice matrix) of XSRM, there is no need to traverse all objects in $Tidsets\,(X_i)$ and $Tidsets\,(X_j)$. We just need to compare the objects in the same row of $Tidsets\,(X_i)$ and $Tidsets\,(X_j)$. If they are equal, then put them in the result intersection $Tidsets\,(R)$.

(ii) Our scheme draws on the idea of matrix dividing, and adds a priori constraint into it to further optimize the association analysis algorithm.

As the program is running, XSRM is generated continuously. And consecutive parts of XSRM generated in a certain time can be taken out for covert channel detection adequately. So it is very suitable for this scheme to use the idea of partitioning, pruning, and recombination with frequent item set mining.

Table 2: Event merging algorithm

Input: XSRM truncated by time window, s_{min}	
Output: Frequent itemset L consisting of all event flows	
1	scan event database to get initial frequent itemset L_1
2	$i = 1,\ j = 2,\ n = 1$
3	$EventMerge\,(L_n)$:
4	for $X_i \in L_1$ do
5	for $X_j \in L_1$ && $j > i$ do
6	generate new candidate itemsets $R = X_i \cup X_j$
7	$Tidsets\,(R) = Tidset\,(X_i) \cap Tidset\,(X_j)$, where $Tidset\,(X_i)$ and $Tidset\,(X_j)$ are colums $A\,(:, i)$ and $A\,(:, j)$ in truncated XSRM
8	if $\lvert Tidset\,(R)\rvert \geqslant s_{min}$ then $i = i + 1,\ j = j + 1$, else break;
9	$L_{n+1} = L_n \cap R$
10	if $n \leqslant k$, then $EventMerge\,(L_{n+1})$

Practical operations are as follows. We analyze each small scale XSRMs during a certain period of time (such as every 10s), rather than wait for a long time to generate a whole large XSRM. However, the partitioned frequent itemsets generated in the XSRM within a certain period of time are not totally necessary for global analysis. In other words, they contain some noisy data. Therefore, we learned from the classical Apriori algorithm, using a priori nature in our algorithm, add in a pruning step to remove some useless candidate frequency itemsets. After mining some of small XSRMs with the threshold minimum support, we cut the frequent itemsets obtained from the next XSRM according to the previous ones. Deleted objects are those that have not occurred in the previous blocks. Ultimately, we obtained the global results by merging all local frequent itemsets.

According to our algorithm, firstly, we scan the same the events within a predetermined time window, counting the itemset number

which is denoted as q. Then calculate the intersection of the frequent n itemsets and do the trimming for generating frequent $n+1$ itemsets. Iterate those steps until only one itemset remains. When determining the minimum confidence threshold c_{min}, where $c_{min} = k/q$, we can set an appropriately high c_{min} to reduce the false negative rate. Meanwhile, in terms of detailed matching rules for covert channels, minimum support threshold s_{min} can be suitably large to reduce the false positive rate. The procedures are shown in Algorithm 2.

4.4.2 Rule construction

After merging the suspicious events, we need proceed our exploitation detection manually by using various analysis approaches. However, manual analysis will cause out of service, which is not allowed in cloud business model. Accordingly, automatic detection must be included in our solution. To achieve that goal, some definitions are given as follows.

Definition 6 Sub-rule

Each sub-rule is written in accordance with the format, and represents a restriction for some attributes. The BNF-like Form of a sub-rule is as follows:

sub-rule = ruleHeader("ruleOptions");
ruleHeader = action type Ph Pl;
action = ("pass","alert","activatedBy","revert");
*ruleOption = (keywords":"StringValue";") *;*
keywords = (general, attribute, processing);
*general = (metadata, VMid, Pid, reference *);*
attribute = (CPULoad, memoryAccess, shareMemoryRequest, others);
processing = (countFunc, frequencyCalcFunc, expectationCalcFunc, entropyCalcFunc, others);

Definition 7 Matching Rule

$rule_i = \{ subrule_{i,1} \cap subrule_{i,2} \cap ... \cap subrule_{i,n} \}$

Definition 8 Rules Set

$Rules = \{ rule_1, rule_2, ..., rule_n \}$

4.4.3 Event flow matching

The procedure of event flow matching comprises a series of steps of calculating the attributes in each event and matching the rules with the result (in Figure 3). After acquiring the concise event flow data, we made use of well-defined rules to match the attribute keywords. A covert channel may correspond to multiple rules. Thus, when rules are matched and triggered, the rule-matching module (detailed in Section 5.3) generates a corresponding alarm.

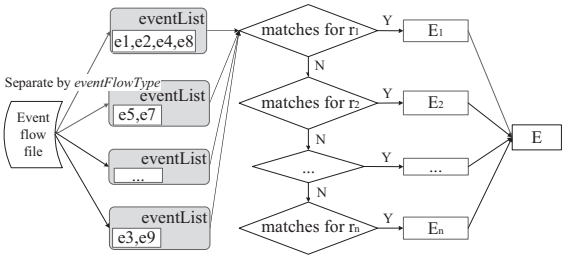

Figure 3: Event flow matching procedures

When matched, the assertion consists of two parts: the conditions (referred to them as LHS, left-hand side) and conclusions (referred to them as RHS, right-hand side). The matching algorithm 3 for detection is as follows:

Table 3: Event flow matching algorithm

Input: event flow file, Rules	
Output: Threat Event Set E	
1	scan the event flow file and separate all event flows into different sets by $eventFlowType$
2	calculate the total number n of all the event flow sets
3	$i = 1$
4	for $\forall rule \in Rules$, do
5	for $subrule$ in a rule, do
6	for $eventList$ in an event flow set, do
7	for $\forall e \in eventList$, do
8	$\forall attribute$ in e, if $LHS\,(attribute) == RHS$ then add e into threat event set E_i
9	$i = i + 1$
10	if $i > n$, then $E = E_1 \cap E_2 \cap ... \cap E_i$, break;

5 Implementation

Based on the above design, we implemented a prototype system on Xen. And we used a open source cloud platform - OpenStack - to manage virtual machines. The prototype consists of five attachable modules: the event logging module, the host information gathering module, the file cleansing module, the event merging module, and the rule-matching module to the covert channel. The logging and monitoring module exists in the VMM level while the others exist in the user space of Dom0.

We made these modules into the plug-ins, and they are deployed on our customized OpenStack. Through this way we can effectively reduce the software execution time. Deployment and management also becomes easier, and users can use different drivers based on their own needs, or disable specific features by shutting down certain plug-ins.

This also gives cloud developers a way to extend the system, by means of the plug-in implementation hook. Developers can define the system extension point in advance, and new backend logic. And the use of this mechanism to expand the system is completely indirect, so we do not need to modify the source code of original system. Moreover, plug-ins can be developed and published separately from the cloud system. Here are some key modules.

5.1 Event logging module

The implementation of the monitoring and logging module is based on both the paravirtualization and the HVM infrastructure of Xen (in Figure 4). By default, guest VM in commodity clouds is always equipped with PV driver no matter whether it is a HVM. So we do not distinguish the ways how a VM implement its event logging module. This module is used to record the deterministic and nondeterministic events in the systems. Precisely, it is responsible for parsing all the commands of guest-VMs which perform the operations such as initialization, startup, and halt. And it is also responsible for recording the key value of the performance counter and gathering the target interrupt events.

The event logging module creates a virtual device. As a character device in Linux, it owns universal system call interfaces such as read, write, IOCTL, etc. This device is actually an agent used to transmit data and commands between user space and the Xen hypervisor, which means all kinds of data will be cached in it. In addition, the device is in charge of registering virtual interrupts callback function and binding the relevant interfaces to read the DomU's information in Xenstore. (The information includes the event channel ID and the reference of the shared ring.)

This module also exports standardized Xen Hypercall interfaces to user space. The daemon of the interfaces accepts the virtual interrupt transferred from the log processing unit and calls the callback functions to read the log data from the virtual device mentioned above. The daemon will use multithreading in implementation to process the users' commands and read the log in parallel.

If the VM is just a HVM without a PV driver, to perform event acquisition we must intercept a variety of events during the VM Exit/VM Entry interval. Those events including hypercalls, I/O requests, external interrupts and various exceptions all can be captured from the current Virtual Machine Control Structure (VMCS). More specifically, Xen provides the trapping and emulating functions to interfere VM's behaviors by set the specific bitmap of VMCS. Therefore, we leverages and modifies those functions to achieve our goal. For instance, to intercept a hypercall, we just overloads the dedicated function which is used for Xen handling the DomU's VMCALL instruction.

It is necessary to specify how to intercept a guest-VM's exceptions. When the guest-OS executes the INT command it must access the IDT table. In this paper, the method we used to intercept exceptions is by setting the limit value of the IDT. When the processor executes the INT command, the limit value of the IDT will be checked first, to see whether the vector is read by the processor. If it is exceeded, the #GP exception is generated. Consequently we only need to set a relatively smaller limit value, to force the guest-OS to cause a #GP abnormal VM Exit for follow-up analysis.

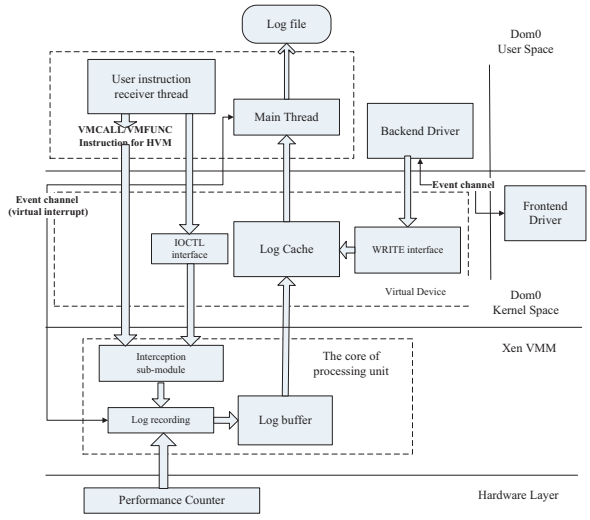

Figure 4: The structure of event logging module

5.2 Host information gathering module

All operations in information gathering module are based on cloud platforms, rather than on guest-VM. That makes the scheme more global and more transparent, and suitable for the application environment.

The information gathering module collects data based on VMI mechanism with some APIs such as libVMI [40] and LibVirt [41]. For example, we used XenAccess to learn from some information about the hardware environment, and we captured all guest-VMs' process lists by using libVMI. By using those VM Introspection tools, we can access the internal data of guest-VMs without their knowledge of VMM's interference.

The collected data will be unified in designed XML format. Meanwhile, by using crawler script, host information gathering module obtains the log files and the security configuration files from the dedicated node in cloud platforms (such as OpenStack controller node, VMWare's vCenter and IBM pSeries' HMC).

The information (the metadata attribute of the event) to be gathered consists of two parts: firstly, changes in guest-VM's log system and configurations, such as the modification to the firewall rules, the addition/deletion/modification of the user's privilege, and whether the service is opening or close. Secondly, the changes in hardware environment, such as the CPU load of the platform, the average access time to cache and memory, and the intervention to the skb-related process (in network devices).

5.3 Rule-matching module

The main goal of rule-matching module is to process the attributes and to match the event flow with the predefined rules.

The initial rules are made according to the specific features of the covert channel, and new rules can be established by administrators, users, expert systems, or knowledge repositories. With the warning feedback mechanism, the rule-matching module can modify the rules and add the improved rules to the rule base.

The policy-script Engine is the core of the rule-matching module, and the Engine's core is the policy-script Interpreter and the attribute-specific Analyzers. The policy-script Interpreter is used to parse the regular expressions in the rules; the attribute-specific Analyzers

invoked by the attribute-specific handler will match the formative events with the rule sequences in the rule base, and decide what to do next after comparison. Once the engine alerts, it will export the corresponding underlying covert channel information as a warning and suggest the users to repair it. By the way, we introduced two automatic sub-detection engine (detailed in Section 6), among which, one can detect the covert channels based on CPU, Cache, or Memory bus, while another can detect the covert channels based on shared memory.

6 Evaluation

We performed the evaluation in the following ways: verifying the correctness and effectiveness of our method, and measuring the performance penalty of our implementation.

6.1 Testbed

Our testbed includes a Sugon TC4600 rack-mounted server with ten blades which each has two 2.50 GHz Intel Xeon E5-2670 V2 CPU and16*8 GB of RAM.

We established OpenStack as the open source cloud computing test platform on our testbed, and deployed Xen, VMWare and Hyper-V on each compute nodes. We started up three virtual machines on each physical host and their OS are Windows XP, Ubuntu 12.04 LTS, and MintOS 13.

6.2 Correctness and effectiveness verification

6.2.1 Sample covert channel exploitations and experiment result

We implemented three practical and malicious channel exploitations as samples to examine our framework.

A. Traditional cache-based covert channels hidden in hyper-threading systems can perform well, and have a high bandwidth. However, in a virtual environment, their transmission rate is far less than before. That is the consequence of the time uncertainty caused by virtual address translation. Therefore, if you want to transfer information, you have to use a special access violation method. Wu [33] found an approach to bring about a kind of special memory bus deadlocks, which will lead to a global access conflict. Memory access latency appears, which can be used to construct covert channels. Likewise, we designed a special atomic operation to make a covert channel, too. When the malicious process accesses two unaligned cache lines, the atomic operation on cache lines can only be ensured by the lock of memory bus, but not the lock of cache. In this case, all the virtual machines on the physical platform cannot access memory, and then malicious Receiver knows that the Sender is passing on a message.

B. Page copy is one of shared memory mechanisms in Xen. Sensitive information can be hidden in the parameter of the page copy commands [35]. It is achieved by assigning the structure `hvm_grant_table_op_t` to the Hypercall `hvm_grant_table_op`. The structure contains required parameters for the page copy function. The grant reference or pseudo physical addresses are designated by the system, and cannot be tampered by users. However, the two operands, offset and the data length, are specified by the user. In that way, attackers can hide the sensitive information into these two parameter fields in covert communication while the page copy is legitimate through public channel. Like this, malicious virtual machines could leak sensitive information furtively. For example, when malicious sender VM notifies the receiver VM that the offset address of subpages in entire page is 0x0111, the receiver can get a binary bit string “000100010001”.

C. In virtualization environment, physical CPU cores are assigned to each user's processes by vCPU scheduling. However, it is easy to be exploited by attackers in covert communication [4]. The scheduler in physical machine usually adopts the 'credit' algorithm to assign the value 'weight' and the value 'cap' (the upper limit of vCPU's available time) to each vCPU in accordance with their security levels. But when two vCPUs have the same weight and cap value, it is possible to build a covert channel. For example, due to the scheduler algorithm, both the sender and the receiver's tasks will be divided into several time periods. Each domain performs the task with no coherence. When one domain is executing its tasks, the physical CPU is assigned to the vCPU which belongs to the domain. Meanwhile, other domains are at rest. So in other words, when the sender domain yields the CPU at the time of rest, the receiver domain is notified. This is the way to use CPU load to form a message. The specific exploitation procedures are as follows. Firstly, the sender executes some loop script, and inserting some intervals every once in a while; Then the receiver confirms that the increase in the CPU load which is derived from the sender. After the receiver confirms the CPU load increase, it can transfer the bit to some spyware outside the cloud and sleep for a short while.

6.2.2 Example sub-detection engines

For identifying some generic covert channels in cloud environment, we realized three typical sub-detection engines on the basis of the framework proposed.

Sub-detection Engine 1: Detecting timing channels based on cache or memory bus contention.

Covert channel exploitations based on cache or memory bus usually go with abnormal memory access events. Firstly, the cache load will be much higher than normal value when such covert channels are active. At the same time, we can easily get abnormal memory access events from the logging and monitoring module, even some cloud platforms (such as vCenter) have already provided a blocking parameter that counts the memory bus contentions and systematic blocking times. So it can be identified as a potential threat when these special memory access operations are more than the threshold value during a certain time.

Example rule: *alert MemoryCC allEventNode (memoryBlockNumber* $\geqslant$ *threshold) AND alert CacheCC allEventNode (CachePayloadExpectation* $\gg$ *noiseThreshold)*

Sub-detection Engine 2: Detecting storage channels based on sharing memory.

Covert channels based on sharing memory often transmit data by modifying parameters of the authorized operations. We can find potential covert channels by surveying operations of authorized access and page copy. Observing whether the request memory sharing operations happened too frequently, extracting offset and data length, and comparing the sharing pages with invoking command could check if there is a storage channel exploitation. Because the length of the sub-page must be less than or equal to the difference between length of the page and initial address of the sub-page, thus the length of the variable as grant operation parameter is not able to be too long. Therefore, the quantum of covert data is extremely tiny in a one-time transfer, and that is why the request operation is frequent. So we can define a rule as follows.

Example rule: *alert inter-VMCC from VMid to VMid (gnttabOP.lenth* $\geqslant$ *pagesize) AND alert MemoryCC allEventNode (times of shareMemoryRequest* $\geqslant$ *threshold value)*

Sub-detection Engine 3: Detecting timing channels based on vCPU scheduling and CPU loads.

Covert channel based on CPU loads must be built on the same CPU core. The sender and

Table 4: Usefulness of specific attributes in detecting covert channels

Detecting \ Attributes	Cache hit	Memory access	CPU load	Shared memory Req	Params of gnttab
Covert channel A	✓	✓			
Covert channel B		✓		✓	✓
Covert channel C			✓		

Table 5: Overall detection results

	TP	TN	FP	FN
1	587	30	0	13
2	590	29	1	10
3	590	29	1	10
4	592	30	0	8
5	597	28	2	3
Average	590.2	29	1	9.8
Rate	0.9837	0.9667	0.0333	0.0163

the receiver transmit bit flow by executing program loop repeatedly and measuring the execution running time. When stealing the information, the attacker has no need to consider the high CPU occupation ratio. But from the perspective of an external observer, when executing program loops, the sender also occupies almost full load of one CPU core. Therefore, we can establish our detection based on monitoring CPU loads.

We can set a threshold, for example, 75%. When the oc-cupation rate is more than the threshold value, we can record it as '1'. On the other hand, if the rate is less than 75%, we record it as '0'. From this, we can get a series of numbers by monitoring CPU utilization continuously. If there is a regular change of this string and at the same time we actually do not run any program, then we can decide that our virtual machine suffered an attack. Finally, we can locate the relevant processes and ring the alarm promptly.

Example rule: *alert inter-VMCC from VMid to VMid (vCPU-id.AffinityCPU = vCPUid.AffinityCPU AND Operation = alternate) AND alert CPUCC allEventNode (cpuLoad ⩾ threshold)*

6.2.3 Test result and analysis

We designed various rules for sub-detection engines, and set the time window (TW) as 10s. Malicious sender's program was deployed on MintOS while sub-detection engines are deployed on Ubuntu with Xen hypervisor. To help lower error rates, we encode each signal of covert data as opposed to the entire bit string. This localizes the effect of bit errors (i.e. channel noise causing an erroneous change in signal assertion). To observe the effect and the usefulness of specific attributes such as memory access interval and CPU affinity, we run multiple tests for a long time, as shown in Table 4. After rectifying the threshold value appropriately, alarms and alarm logs of covert channel A and B were successfully attained.

Table 5 shows that in the five groups of experiments, 630 samples (600 normal communication samples, 30 covert channel samples) were repeated 5 times. The average detection error number is 9.8, average missing number is 1.0. It is obvious that the false positive rate is less than five percent, and false negative rate is extremely trivial, almost less than two percent. In other words, our detection scheme can find almost all the behaviors of suspicious covert channels. Hence we mainly focus on the false alarm rate.

Figure 5 and Figure 6 show the false positive rate of the detection for covert channel exploitations A and B after each sub-rule is added into the sub-engines. The above-mentioned VM event log maintains much useful data about the hardware status, such as CPU load, cache hit/missing rate, and memory ac-

Table 6: Comparison of different detection schemes

Methods \ Channel types	CPU load-based	Cache based	Shared memory-based	Others
Active traffic analysis [19]				✓
C2Detection [21]			✓	
Detection with TDR [23]	✓	✓		
BusMornitor [42]		✓		
Our framework	✓	✓	✓	✓

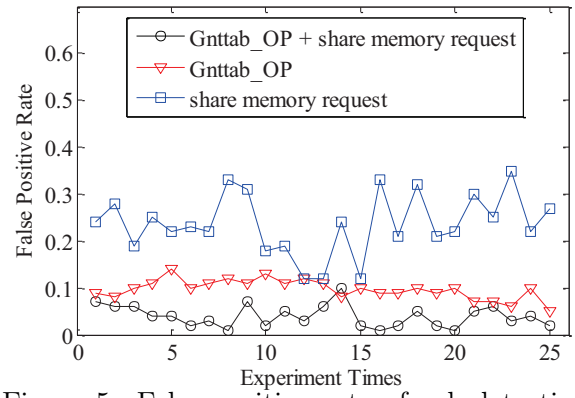

Figure 5: False positive rate of sub-detection engine 1

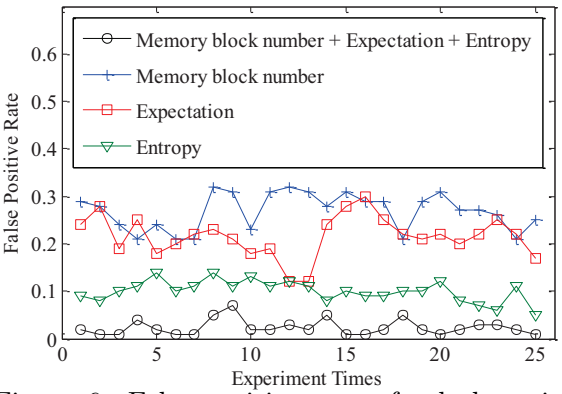

Figure 6: False positive rate of sub-detection engine 2

cess frequency. That is because all the actions in the upper layer can be mapped to the underlying hardware event. In addition, we can get all shared memory request and Parameters of all hypercalls from logs of Xenstore and logs obtained by LibVirt API.

Every time an attribute is mixed in, the false positive rate will decrease significantly. Thus, as long as setting appropriate rules, the more detailed rules are, the better the result is. Furthermore, it indicates that our scheme is effective by inducing event association analysis to detect the attacker's behavior.

The experiments also show that the biggest advantage of this scheme compared to other single detection schemes is that we can detect more types of covert channels. The following Table 6 compares the channel types covered by this scheme and others in cloud or virtualization environments.

6.3 Performance analysis

In order to see the performance overhead of the prototype system designed in this paper, we measured the system resource consumption of the XSRM algorithm and the detection modules.

6.3.1 Performance on our improved event association algorithm based on XSRM

It is obvious that XSRM's sizes are various in different time windows. Figure 7 and Figure 8 show the performance (execution-time of event analysis for XSRMs) of different algorithms at different minimum support (s_{min}). Our algorithm is significantly better than the classical Apriori algorithm, and the processing speed of ours is nearly twice faster than the Eclat algorithm.

When m, the amount of rows in XSRM, approaches 10^5 order of magnitude, the space occupied by the shared resource matrix will reach

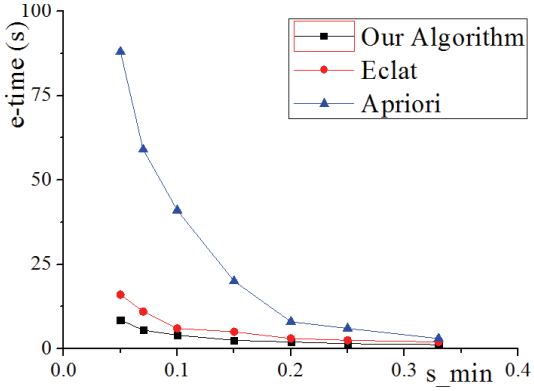

Figure 7: Execution time of three algorithms for XSRMs generated in 5s

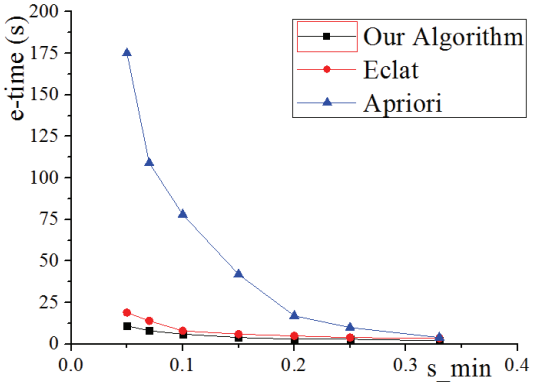

Figure 8: Execution time of three algorithms for XSRMs generated in 15s

nearly MB level. Current memory conditions may not meet the need. So this algorithm does not apply to the data set of very large mining projects. Nevertheless, when the data set is not cumbersome (less that MB magnitude), it is very worthwhile to exchange the large amount of time for relatively cheap computing resources.

6.3.2 Performance of the prototype

In order to test the performance of our scheme, we run the two sub-detection engines respectively. Because the sub-engine 3 itself relies on CPU scheduling and CPU utilization to perform the inspection, so we have no need to test its performance about CPU. More importantly,

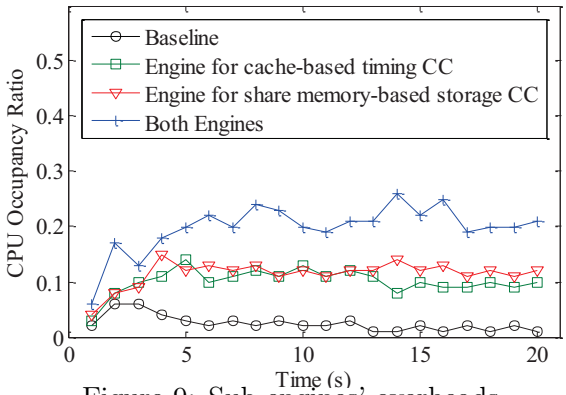

Figure 9: Sub-engines' overheads

CPU usage is not accurate enough when a CPU load-based covert channel is running. Figure 9 indicates the Xen dom0's CPU load in different situations. Every average is for ten runs.

7 Conclusion

Our work focuses on eliminating the potential threat of covert channel in cloud environment (virtual environment), and takes the differences between virtualization infrastructure and traditional computing environment fully into consideration. The method traverses and searches possible covert channels by combining shared resources matrix method with event association analysis.

We put forward a comprehensive and effective covert channel detection method in cloud computing environment. It is novel because our method can identify covert channel exploitations by recognizing their behaviors. In addition, previous studies mostly aimed at unique channel. In contrast, our framework can do well in detecting most covert channels in cloud from the perspective of resources sharing.

An algorithm based on shared resources matrix method and event association analysis is proposed to uncover the covert channel by sniffing attackers' intents. Because covert channels are always well-constructed, utilizing this

method into detection can fits the practical situation better. Moreover, we proved that the more we introduce appropriate attributes, the better our method can do.

At the same time, our work is a first step towards using event association analysis to detect covert channel in cloud computing environment. We improved the original shared resources matrix algorithm, and used XSRM for preprocessing the event logs and security configuration files. It can reduce the loads of constructing a huge shared resources matrix. Differentiating the target device on which events happened and providing important event information for analysis are also able to prevent administrators from being overwhelmed by a great mass of false alarms.

Traditional covert channel detection schemes always rely on human intervention. In order to avoid the tedious manual analysis and the state space explosion, information flow analysis method is not used in the proposed automatic framework. The programs can run on a cloud platform in real time with few overheads, and can assess the state of cloud environment transparently. In addition, our solution is deployed on the cloud platform as plug-ins, which makes the cloud more scalable and easier to manage.

Acknowledgement

This work was partially supported by the National High-tech R&D Program of China ("863" Program) (Grant No. 2015AA016004), the National Natural Science Foundation of China (Grant No. U1536204, 61373169, 61303213), the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD) and Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology (CICAEET).

References

[1] Zissis D, Lekkas D. Addressing cloud computing security issues. *Future Generation computer systems* 2012; **28**(3):583–592.

[2] Jin H, Xiang G, Zou D, Wu S, Zhao F, Li M, Zheng W. A vmm-based intrusion prevention system in cloud computing environment. *The Journal of Supercomputing* 2013; **66**(3):1133–1151.

[3] Bellovin SM. Virtual machines, virtual security? *Communications of the ACM* 2006; **49**(10):104.

[4] Okamura K, Oyama Y. Load-based covert channels between xen virtual machines. *Proceedings of the 2010 ACM Symposium on Applied Computing*, ACM, 2010; 173–180.

[5] Salaün M. Practical overview of a xen covert channel. *Journal in computer virology* 2010; **6**(4):317–328.

[6] Ristenpart T, Tromer E, Shacham H, Savage S. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. *Proceedings of the 16th ACM conference on Computer and communications security*, ACM, 2009; 199–212.

[7] Zhang Y, Juels A, Reiter MK, Ristenpart T. Cross-vm side channels and their use to extract private keys. *Proceedings of the 2012 ACM conference on Computer and communications security*, ACM, 2012; 305–316.

[8] Zhang Y, Juels A, Oprea A, Reiter MK. Homealone: Co-residency detection in the cloud via side-channel analysis. *security and Privacy (SP), 2011 IEEE Symposium on*, IEEE, 2011; 313–328.

[9] Kim T, Peinado M, Mainar-Ruiz G. Stealthmem: system-level protection against cache-based side channel attacks in the cloud. *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, 2012; 189–204.

[10] Varadarajan V, Ristenpart T, Swift M. Scheduler-based defenses against cross-vm side-channels. *23rd USENIX Security Symposium (USENIX Security 14)*, 2014; 687–702.

[11] Moon SJ, Sekar V, Reiter MK. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2015; 1595–1606.

[12] Lampson BW. A note on the confinement problem. *Communications of the ACM* 1973; **16**(10):613–615.

[13] Lipner SB. A comment on the confinement problem. *ACM SIGOPS Operating Systems Review*, vol. 9, ACM, 1975; 192–196.

[14] Volpano D, Irvine C, Smith G. A sound type system for secure flow analysis. *Journal of computer security* 1996; **4**(2-3):167–187.

[15] Haigh JT, Kemmerer RA, McHugh J, Young WD. An experience using two covert channel analysis techniques on a real system design. *Software Engineering, IEEE Transactions on* 1987; (2):157–168.

[16] Kemmerer RA. Shared resource matrix methodology: An approach to identifying storage and timing channels. *ACM Transactions on Computer Systems (TOCS)* 1983; **1**(3):256–277.

[17] Tsai CR, Gligor VD, Chandersekaran CS. On the identification of covert storage channels in secure systems. *Software Engineering, IEEE Transactions on* 1990; **16**(6):569–580.

[18] Qing S, Shen C. Design of secure operating systems with high security levels. *Science in China Series F: Information Sciences* 2007; **50**(3):399–418.

[19] Bates A, Mood B, Pletcher J, Pruse H, Valafar M, Butler K. Detecting co-residency with active traffic analysis techniques. *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, ACM, 2012; 1–12.

[20] Varadarajan V, Zhang Y, Ristenpart T, Swift M. A placement vulnerability study in multi-tenant public clouds. *24th USENIX Security Symposium (USENIX Security 15)*, 2015; 913–928.

[21] Wu J, Ding L, Wu Y, Min-Allah N, Khan SU, Wang Y. C2detector: a covert channel detection framework in cloud computing. *Security and Communication Networks* 2014; **7**(3):544–557.

[22] Dunlap GW, King ST, Cinar S, Basrai MA, Chen PM. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. *ACM SIGOPS Operating Systems Review* 2002; **36**(SI):211–224.

[23] Chen A, Moore WB, Xiao H, Haeberlen A, Phan LTX, Sherr M, Zhou W. Detecting covert timing channels with time-deterministic replay. *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014; 541–554.

[24] Aviram A, Hu S, Ford B, Gummadi R. Determinating timing channels in compute clouds. *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, ACM, 2010; 103–108.

[25] Wray JC. An analysis of covert timing channels. *Journal of Computer Security* 1992; **1**(3-4):219–232.

[26] Li P, Gao D, Reiter MK. Stopwatch: a cloud architecture for timing channel mitigation. *ACM Transactions on Information and System Security (TISSEC)* 2014; **17**(2):8.

[27] Askarov A, Zhang D, Myers AC. Predictive black-box mitigation of timing channels. *Proceedings of the 17th ACM conference on Computer and communications security*, ACM, 2010; 297–307.

[28] Zhang D, Askarov A, Myers AC. Predictive mitigation of timing channels in interactive systems. *Proceedings of the 18th ACM conference on Computer and communications security*, ACM, 2011; 563–574.

[29] Lina W, Han-jun G, Wei L, Yang P. Detecting and managing hidden process via hypervisor. *Journal of Computer Research and Development* 2011; **48**(8):1534–1541.

[30] Tan Y, Nguyen H, Shen Z, Gu X, Venkatramani C, Rajan D. Prepare: Predictive performance anomaly prevention for virtualized

cloud systems. *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, IEEE, 2012; 285–294.

[31] Zhai J, Liu G, Dai Y. Detection of tcp covert channel based on markov model. *Telecommunication Systems* 2013; **54**(3):333–343.

[32] Kadloor S, Kiyavash N, Venkitasubramaniam P. Mitigating timing based information leakage in shared schedulers. *INFOCOM, 2012 Proceedings IEEE*, IEEE, 2012; 1044–1052.

[33] Wu Z, Xu Z, Wang H. Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud. *IEEE/ACM Transactions on Networking (TON)* 2015; **23**(2):603–614.

[34] Wu J, Ding L, Wang Y, Han W. Identification and evaluation of sharing memory covert timing channel in xen virtual machines. *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, IEEE, 2011; 283–291.

[35] Gao H, Wang L, Liu W, Peng Y, Zhang H. Preventing secret data leakage from foreign mappings in virtual machines. *Security and Privacy in Communication Networks*. Springer, 2011; 436–445.

[36] Hovhannisyan H, Lu K, Yang R, Qi W, Wang J, Wen M. A novel deduplication-based covert channel in cloud storage service. *2015 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2015; 1–6.

[37] Garfinkel T, Rosenblum M, *et al.*. A virtual machine introspection based architecture for intrusion detection. *NDSS*, vol. 3, 2003; 191–206.

[38] Liang J, Zhao X, Li D, Cao F, Dang C. Determining the number of clusters using information entropy for mixed data. *Pattern Recognition* 2012; **45**(6):2251–2265.

[39] Lee W, Stolfo SJ, *et al.*. Data mining approaches for intrusion detection. *Usenix security*, 1998.

[40] libVMI API. http://libvmi.com/api.

[41] libvirt virtualization API. http://libvirt.org/html/libvirt-libvirt-event.html.

[42] Saltaformaggio B, Xu D, Zhang X. Busmonitor: A hypervisor-based solution for memory bus covert channels. *Proceedings of EuroSec* 2013; doi:10.1.1.299.2497.