CPU LOAD BASED THREAT PREVENTION FRAMEWORK FOR COVERT CHANNEL ATTACKS

Kunchala Little Flower* ,Research Scholar, CSE department at Sri Satya Sai University of Technology & Medical Science-Sehore, MP.

Dr.Neeraj Sharma, Associate Professor Department of CSE- at Sri Satya Sai University of Technology & Medical Science-Sehore, MP.

Abstract: Numerous virtual machines on a solitary virtual machine mon-itor are segregated from one another. A malignant client on one virtual machine ordinarily can't hand-off restricted information to other virtual machines without utilizing unequivocal correspondence media like shared les or an organization. In any case, this detachment is undermined by correspondence in which CPU load is utilized as a secretive channel. Sadly, this danger has not been completely perceived or assessed. In this review, we quanti-tatively assess the danger of CPU-based secret channels between virtual machines on the Xen hypervisor. We have created CCCV, a framework that makes a clandestine channel and conveys information subtly utilizing CPU loads. CCCV comprises of two client processes, a sender and a beneficiary. The sender runs on one virtual machine, and the beneficiary sudden spikes in demand for one more virtual machine on the equivalent hypervisor. We measured the transfer speed and correspondence precision of the incognito channel. CCCV imparted 64-bit information with a 100% achievement rate in an optimal climate, and with a success pace of more than 90% in a climate where Web servers are handling demands on other virtual machines.

Keywords:Covert channels, virtual machine monitors, Xen hypervisor.

1. INTRODUCTION:

A critical advantage of VMM-based virtual working with is strong separation.

Instinctually, a VMM gives customers a double dealing of monopolizing a completely set up PC. VMM-based host-ing outfits each customer with a lot of virtual hardware that is unequivocally disengaged from that of others. In this kind of working with, unquestionable events of working systems are run specifically en-vironments (virtual machines), and hereafter, wise resources, for instance, les are not split between different conditions. That isolation feature is unequivocally required while working with the states of regularly untrusting customers.

VMM-based working with is particularly engaging for individuals who should store and control special data in a rented environment, because different virtual conditions regularly don't share any real resources. Thusly, in-line spillage can in a general sense be hindered by checking network correspondence, as by virtue of an environment dependent on a dedicated genuine machine. On the off chance that a malignant steady of gram on one virtual machine attempts to send special data to an external environment (counting another virtual machine

worked with on a comparative genuine machine), network noticing programming, for instance, [12, 15, 16, 17] can without a doubt distinguish and prevent the correspondence.

Lamentably, as has been raised before [7, 9], conspiring tasks can talk with each other through a channel that isn't gotten ready for correspondence by director istrators. The channel is habitually called an undercover channel. A wide extent of resources are alluded to be usable as covert channels, e.g., TCP headers [2], timing of estimation [18], and memory access lethargy [3, 4, 10]. A noticing structure that investigates simply normal channels forgets to see information spillage through surreptitious channels. In VMM-based hosting, malignant ventures can make cryptic channels between dierent virtual machines [11].

Maybe the most favorable resources for clandestine channels between virtual machine is CPU load, which can be approximated by the proportion of time taken for certain computations. Since CPUs or focuses are routinely split between dierent virtual machines and cycles, a malevolent cooperation (e.g., spyware) on one virtual machine can secretly talk with its companion on another virtual machine by changing the CPU weight and making the buddy see the change. On the other hand, dinary security structures can't recognize or thwart the communication. Focal processor load is an invaluable resource for aggressors considering the way that changing the CPU load doesn't require administratop benefits, and it is less difficult to control in a period fragile way than most various pointers, for instance, page weakness rates.

In this work, we tend to foster CCCV (Covert Channels using CPU masses between Virtual machines), associate concealed channel correspondence framework between virtual machines on the Xen hypervisor [1]. It empowers processes in DomU virtual machines to talk with one associateother| each other utilizing an actual CPU because the correspondence medium. CCCV makes favorable to cesses on dierent virtual machines convey through knowledge on the execution seasons of very little program code. we tend to quantitatively assess the capability of CCCV; in particu-lar, we tend to justify its knowledge transmission and truth. it's disbursed while not adjusting the code of the Xen hypervisor or a traveller OS bit. searching outcomes show that CCCV cycles will convey at zero.49 rate with 100 percent exactness in associate optimum climate and at the same rate with around ninety one truth in any event, once different virtual machines ar facilitating a creditably stacked net server on the equivalent hypervisor.

This review centersaround CPU load and does not contemplate the assembly of hush-hush channels with completely different assets, for instance, operation immobility during this work. we will work on the transmission capability and truth of CCCV by change of integrity CPU load (execution time) knowledge with knowledge concerning completely different assets. withal, this can be left for future work, associated here we tend to specialise in recognizing a boundary of the transfer speed and exactness of an concealed channel framework that controls and notices execution time knowledge alone.

This paper portrays a basic idea for loosening up CCCV to work honorably moreover when dierent virtual machines are not by and large intended to comparable game plan of focuses. This paper further shows starter exploratory results for surveying the increase. Before rushing into assembling exploratory results in complex conditions, regardless, we focused in on getting achieves the direct situation above. The results will be significant as urgent data for future assessment.

This paper is composed as follows. Portion 2 portrays CPU schedulers in Xen, and Section 3 explains the implementation of the proposed system, CCCV. Portion 4 shows our preliminary outcomes, and Section 5 differences our work and related work. Section 6 wraps up this paper with a compact summary and the portrayal of a couple of future undertakings.

Two region schedulers are available in the current variation of Xen: the SEDF scheduler and the credit scheduler. Since the credit scheduler is standard in late structures, CCCV expects its use.

2. VIRTUAL MACHINES ON THE XEN HYPERVISOR

The credit scheduler sporadically consigns credits (CPU time considered use) to each VCPU. The CPU time actually gobbled up by a VCPU is deducted from its credits. The chief designates a weight and a cap to each do-essential, and the credit scheduler concludes the credits given to each VCPU as shown by these two characteristics. Naturally, the weight tends to the need of a space. Credits are accomplice appropriated corresponding to the greatness of the spaces. The cap tends to a greatest cut off on the CPU time consumed by a VCPU. The default worth of a cap is zero, which infers that there is no limitation on CPU time. The state of a VCPU that has eaten up all of its credits is brought over, and the other state is called under. The credit scheduler schedules VCPUs with the under state helpfully. VCPUs with the all through state get CPU time exactly when all VCPUs with the under state are inert.

3. PROPOSED METHOD

3.1 THREAT MODEL

We acknowledge that an assailant has a strategy for imbuing spyware into a space administered by another person. A guest OS in the space stores secrets, for instance, passwords or Mastercard numbers. We consider this space a sender region. The individual or the VMM manager hopes to thwart infor-mation spillage by introducing network noticing programming (e.g., [12, 15, 16, 17]) or a well congured re wall in the dobasic, or by building a virtual private association that isolates the space from the Internet. Spyware in the space can-not spill advantaged experiences just by sending them through the Inter-net in light of the fact that the noticing programming may recognize the spillage or the rewall or virtual private association may hold the spyware back from getting to the Internet.

We furthermore acknowledge that the trader of the spyware can make or rent a space near the sender region on the identical hypervisor. This is a portion of the time possible just by using the virtual working with expert association that has the sender space. We consider this space a beneficiary region. Since the attacker manages the beneficiary space, the assailant doesn't present any security programming or rewall around here. The attacker executes CCCV in the sender and beneficiary regions (the aggressor makes the spyware contain the sender part of CCCV). The spyware sends secrets to the beneficiary space through a covert channel, and a poisonous program in the recipient region sends the way in to the attackers website page through the Internet.

3.2 CPU Load as a CovertChannel

We assume that a hypervisor is hosting two DomUs, domain α and domain β, and that VCPUs in the domains are mapped to the same CPU (core). Both domains are given the same weight and a cap of zero. CCCV runs a receiver process in domain α and a sender process in domain β. Each process executes task t in its domain. We also assume that t requires four time slices for completion, and no CPU time is consumed by programs other than t.

Figure 1 illustrates the timeline of execution in domain α in diff erent cases. The upper part shows the case in which domain β has no runnable task. In that case, domain α con- sumes most of the physical CPU's time. Consequently, the amount of time that elapses during the execution of t equals four time slices. In contrast, the lower part shows the case in which domain α and domain β start executing t at the same time. Since the domains run on one physical CPU, either of the domains is alternately scheduled in every time slice. One domain must pause when the other is scheduled. Hence, the amount of time that elapses during the execution of t in domain α increases to seven time slices. This means that domain α can know whether domain β is executing some computation by measuring the time required to complete t; domain β can send information to domain α by changing the CPU load according to a communication protocol, such as "bit 1 is being sent if domain β is executing t, and bit 0 is being sent otherwise."

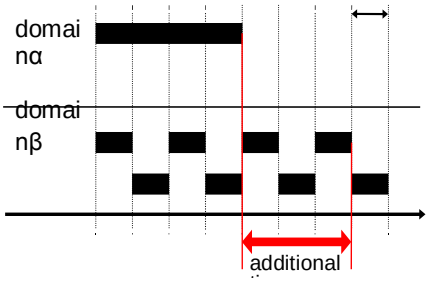

Figure. 1. Change in execution time due to sharing of a physical CPU.

Normally, CPU loads in different areas can upset com-munication by CCCV. We don't guarantee that CCCV can generally accomplish wonderful correspondence. We possibly

guarantee that CCCV functions admirably assuming the CPU heaps of cycles other than the sender and collector don't change signicantly during com-munication.

3.3 CommunicationProtocol

We portray the correspondence convention of CCCV. The sender and beneficiary need to execute unitcomp in a synchro-nized way. Consequently, the two cycles should synchronize the circumstance of correspondence prior to sending or getting a piece stream that addresses insider facts. Also, they should stop intermittently to keep the hidden CPU from be-ing decided as over by the credit scheduler. The current com-munication convention considers these prerequisites. It comprises of three stages: the synchronization stage, the conrmation stage, and the piece transmission stage.

It takes 131.5 s to convey 64 pieces in the current execution of CCCV. The transmission capacity is 64/131.5 = 0.49 pieces per s assuming that no blunder happens. The breakdown of the time is displayed underneath.

Synchronization stage: 2.5 s Conrmation stage: 1 s Stop between conrmation stage and spot transmis-sion stage: 1 s Transmission of pieces: 64 s Stops between transmission of every pieces: 63 s

The correspondence confirmation stage begins after the rest. In this stage, the recipient conrms that the noticed ascent in CPU load is really brought about by a correspondence re-journey from the sender and not by the calculation load because of different cycles. In this stage, the sender stops for a specific timeframe, and the recipient works out how regularly it can execute unitcomp during this period. The number is called standard number, and the period is called standard time interval. As of now, the standard time frame is 1 s. A standard number is utilized in the piece transmission stage. Assuming the recipient distinguishes an ascent in CPU load during the period (i.e., if

the standard number is bigger than a precalculated normal by a limit), it decides that it got a bogus communication demand in the synchronization stage because of CPU use by processes other than the sender. All things considered, the beneficiary re-visitations of the beginning of the synchronization stage. The sender and recipient then, at that point, stop for one more 1 s to set the condition of the fundamental VCPUs as under. Here, a slight hole might in any case exist between the circumstance of the sender and recipient. The length of the hole relies upon the execution season of unitcomp utilized in the synchronization stage. We limit the eect of the hole by giving suf-cient time to correspondence of each piece in the piece transmission stage. To summarize, the CCCV convention requires a sender to make a specic CPU load design prior to communicating pieces of helpful data. This adds to synchronization of commu-nication and prohibition of bogus correspondence and commu-nicationendeavors in mistake inclined conditions.

3.4. Bit Transmission Phase

Then, at that point, the sender and gatherer start the piece correspondence stage, in which they give 64-digit information. The show gobbles up a standard timespan to pass on a tiny

smidgen. The recipient gauges the amount of executions of unit-comp during the standard timespan. Expecting that the number is under 90% of the standard number, it gets bit 1. Other-wise, it gets bit 0. The sender and beneficiary rest for 1 s between transmis-sions of each piece. This rest is familiar with prevent the fundamental VCPU from being chosen as wrapped up. In case there was no rest, an authority would endlessly gobble up CPU time, and consequently the state of the secret VCPU would become over. As depicted more than, a more unobtrusive proportion of CPU time is alloted to an over VCPU.

```
1: { initialization procedure begin }
2: do 100 times
3:     measure the execution time of unitcomp
4: end
5: ave = average of execution times above 6: {
initialization procedure end }
7:
8: { synchronization phase begin }
9: do infinitely
10:    start time = current time
11:    measure the execution time of unitcomp
12:    if the execution time      ≥ ave + 0.03 then
13:       repeat
14:          measure the execution time of unitcomp
15:          end time = current time
16:       until the execution time < ave + 0.03
17:       if end time  –start time  –1.5  ≤ 0 then 18:
             { a higher CPU load observed for about 1.5 s } 19:
             { the CPU load is judged as sender's }
20:          break;
21:       else
22:          { the CPU load is not judged as sender's }
23:          continue;
24: end
25: { synchronization phase end }
26:
27: { confirmation phase begin }
28: ...
```

Figure 3: Algorithm of the receiver process in the initialization procedure and synchronization phase

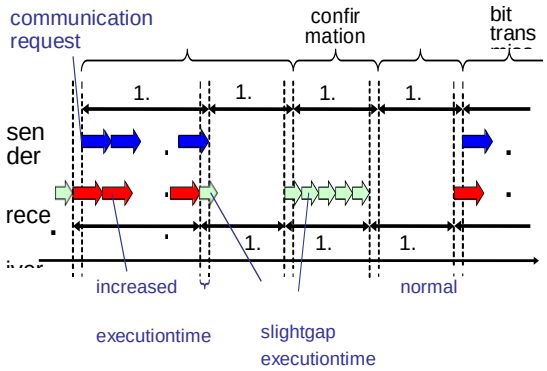

Figure 4: Flow of the CCCV communication protocol

3.5 Expansion to Multi-focus Machines

We portray a fundamental idea for loosening up CCCV to work honorably furthermore when processes in passing on spaces can be wanted to dierent real CPU places. The extensive strong of tocol doesn't acknowledge that VCPUs in correspondence do-mains are continually

arranged significantly. In the extensive show, the sender and the beneficiary make as some strong of cesses as the amount of VCPUs apportioned to a space. We consider the cycles a sender pack and a beneficiary bundle.

All cycles in the sender bundle execute unitcomp in a synchronized manner and measure the breathed easy accord-ing to the principal show. Every association in the recipient assembling furthermore executes a get movement comparably as the principal show. After 64-bit correspondence, how-ever, a cycle in the beneficiary pack attempts to grant the parts of various cycles that start a get movement with a comparative preparation. Then, the cycles conclude a tiny smidgen by a tiny smidgen using a bigger part computation. Pieces are viably sent in the somewhat long show in the event that somewhere near one sender cycle and one beneficiary interaction share a real place during correspondence. Processes in a sender or recipient gathering synchronize with each other using signals and shared mem-ory.

3.6 SECURITY PRESERVATION AND ACCESS CONTROL IN CLOUD COMPUTING

Our security approach enables a data provider to relegate different assurance techniques to different components who approach their data. Right when a sender establishes a connection with few people, the individual is using broadcast encryption, which allows each person to unravel the message using their own private key following receiving the message. The correspondence, on the other hand, can't be unscrambled by anybody outside of the social event. One more time of encryption techniques that rely upon broadcast key organization structures involve only a lone layer of encryption strategy in which the data owner ought to stay dynamic reliably, actually proposed. An encryption approach with two layers was proposed to determine this issue, with the coarse grained encryption done by the data owner to ensure that all cloud-based data is kept secret farthest degree possible.

Most of the encryption calculations referenced give security and keep up with information mystery, subject matter authorities agree. Various methodologies for fine-grained information access control have been proposed, including compulsory access control, optional access control, and job based admittance control techniques, among others. In these models, distinguishing proof is given as interesting names relegated to individuals and assets to indicate it. This strategy is helpful in static frameworks since the arrangement of clients and administrations is known ahead of time, simplifying it to execute. Clients who are dynamic in nature, just as assets that are innately impromptu, are for the most part elements of the frameworks that exist today.

It is a part for securely allocating messages to customer social events, with secret filling in as the fundamental key to advance. A symmetric key K is held by each person from the social event, which is insinuated as the get-together key. Exactly when a message should be shared, it ought to be encoded using K and a short time later shipped off the people from the social event. Since all of the people from the social event have some familiarity with about the letter K, they are generally prepared to unravel and examine the message. Exactly when another

customer joins or leaves a social event, one more assembling key is made and delivered off the get-together people. The messages sent before in the social occasion are not accessible to new people (this is known as in invert secret), and customers who have left the get-together can't gain information from future collaborations in the get-together (forward secret).

4. EXPERIMENTAL RESULTS

Figure depicts the simulation results produced by utilizing the cloud CPABE toolbox for key generation and is based on the findings obtained.

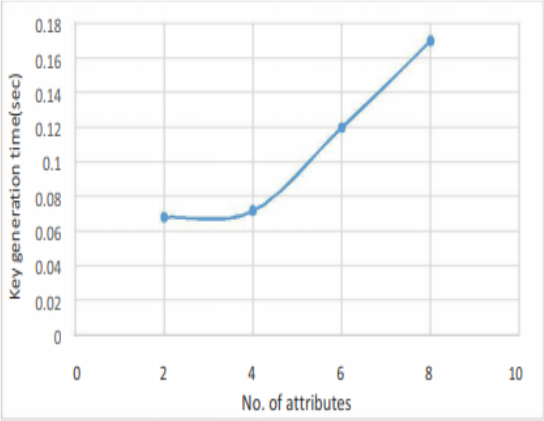

Figure 4.1 Key generation Time

It is clear from the chart that the average time required to create keys in double layer encryption is significantly shorter than the average time required in single layer encryption.

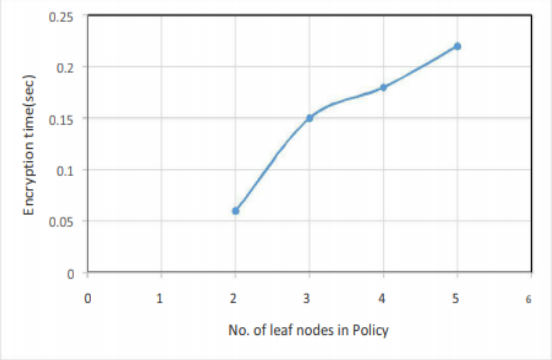

Figure 3.19 Encryption Time

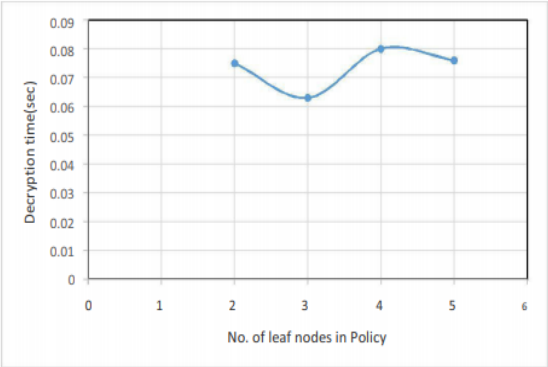

Figure 4.2 Decryption Time

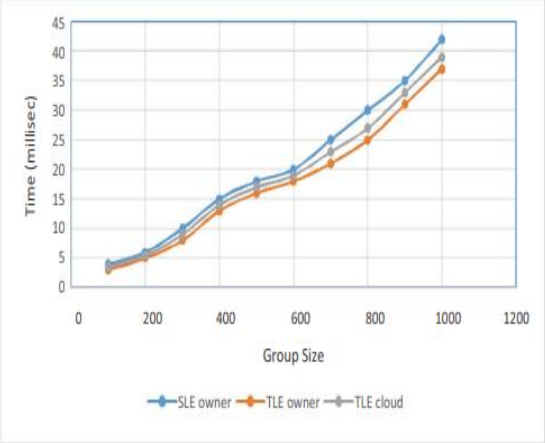

Figure 4.3 Average Time to generate keys for single and two layers

Latest distributed computing patterns, just as the administrations that help them, are intended for moving capacities to outsider suppliers to accomplish more noteworthy adaptability and versatility. Late reviews, nonetheless, have uncovered that information protection and security concerns are the main elements driving associations to move to the cloud. The objective of this paper is to accomplish decentralized twofold encryption, adaptability, and information secrecy for information access control in the cloud while keeping up with information classification.

5. EVALUATION OF COVERTCHANNEL COMMUNICATION BY CCCV:

We estimated the precision under different conditions by changing the general needs of conveying areas and the CPU loads in different spaces that were not identified with CCCV (called outsider areas beneath). We endeavored 64-bit correspondence multiple times for each condition and ex-amined whether synchronization was fruitful and regardless of whether the conveyed pieces were right.

Table 1: Accuracy of communication measured on a single-core machine when no third-party domain was hosted.

S:*R*	3:1	2.5:1	2:1	1.5:1	1:1	1:1.5	1:2	1:2.5	1:3
Success	92	98	100	98	100	61	54	16	4
Sync failure	8	2	0	2	0	39	46	84	96
Bit error	0	0	0	0	0	0	0	0	0

Table 2: Accuracy of communication measured on a multi-core machine when no third-party domain was hosted.

VCPUs in sender domain	2	3	4	8	2	3	1	4
VCPUs in receiver domain	2	3	4	8	3	2	4	1
Succes	0	92	100	2	67	5	7	0
Sync failure	100	8	0	0	33	95	93	100
Bit error	0	0	0	98	0	0	0	0

Table 3: Accuracy of communication measured on a single-core machine when the Apache Webserver was running in two third-partydomains.

CPU load (%)	5-9	8-12	13-17
Success	92	91	74
Sync failure	7	9	13
Bit error	1	0	13

The result is shown in Table 2. A 100% accomplishment rate was cultivated when both the sender and beneficiary space had four VCPUs. That was an ideal case because the two regions were depended upon to put loads on all of the (four) real focuses and between fere with each other. CCCV gained a 92% headway rate when both the sender and gatherer had three VCPUs. In light of everything, synchronization dissatisfaction happened exactly when all of the (three) recipient cycles forgot to see a communication request. Exactly when somewhere around one gatherer processes recog-nized a correspondence interest, they viably got right pieces subject to a larger part computation. Exactly when the sender and beneficiary had two VCPUs, all correspondence attempts failed. This result was ordinary considering the way that the space scheduler was likely going to consign two specific genuine focuses to each space. Exactly when the amount of VCPUs in the sender and the beneficiary were both eight, most correspondence achieved piece bumbles. We acknowledge that the cycle botches were achieved by contention for genuine focuses among processes in a gatherer bundle. Right when the sender and the recipient had a dierent number of VCPUs, accomplishment rates were low.

The results in the ensuing preliminary showed that some synchronization disillusionment and correspondence botches happened when CPU use occurred in various spaces. In any case, when the CPU load was under 12%, CCCV could achieve more than 90% accuracy. Achievement of correspondence depends upon whether the CPU stores of cycles other than the sender and beneficiary vacillatesignicantly during correspondence. Customers of CCCV can restrict the eect of various cycles by choos-ing a fitting time during which the CPU load of various cycles is low (e.g., late around evening time and quickly at the start of the day). In the current show, the recipient adjudicators the sender to bring the CPU load up in any occasion, when the recipients execution isn't immensely degraded. The edge is 90% of the first per-formance. We picked the edge to achieve successful communication regardless, when the recipient has a greater weight (regardless, when the beneficiaries execution times are likely not going to create signicantly).

6. CONCLUSION:

We made and surveyed CCCV, a correspondence sys-tem across Xen virtual machine restricts that uses CPU load as a secret channel. Since CCCV runs as customer processes in guest OSes, the customers shouldn't for a second mess around with administrator benefits on a guest OS or the host OS. We conrmed likely that CCCV could give 64-cycle data in 131.5 s with widely high accuracy. CCCV could examine data with a 100% accomplishment rate in an ideal environment where do-mains other than giving regions were consuming little CPU time. Regardless, when two untouchable spaces were working with a Web server, CCCV gained a headway speed of over 90%. Future assessment could follow a couple of ways. The rst is to coordinate further assessments on various machines, other oper-ating structures, and other virtualization establishment. For example, evaluating secret channels on compartment based sys-tems [6, 14] is a captivating future errand. The second is to perceive how much the information transmission and accuracy of commu-nication by CCCV can be improved. A key is to restrict or level out o varying CPU use by various cycles.

A clear improvement is to introduce overabundance in conferred information (e.g., to send comparable 64-digit data on various occasions with a 10-min length or to take on screw up correction code). The last one is to propose an eective arrangement to prevent secret channel correspondence by CCCV or similar systems. A clear yet uplifting plan is to run a disturbing association that makes a self-assertive CPU load on the identical hypervisor.

1. REFERENCES

[1] *P. Barham, B. Dragovic, K. Fraser, S.Hand,T. Harris, A. Ho, R. Neugebauer, I. Pratt, andA. Warfield. Xen and the Art of Virtualization. In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03), pages 164–177, 2003.*

[2] *T. G. Handel and M. T. S. II. Hiding Data in the OSI NetworkModel.InProceedingsofthe1stInternational Workshop on Information Hiding, volume 1174 of LectureNotesinComputerScience,pages23–38,1996.*

[3] *W.-M. Hu. Reducing Timing Channels with Fuzzy Time. In Proceedings of the 1991 IEEE Symposiumon Research in Security and Privacy, pages 8–20,1991.*

[4] *W.-M.Hu.LatticeSchedulingandCovertChannels. In Proceedings of the 1992 IEEE Symposium on Security and Privacy, pages 52–61,1992.*

[5] *T. Jaeger, R. Sailer, and Y. Sreenivasan.Managing the Risk of Covert Information Flows in Virtual Machine Systems. In Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, pages 81–90,2007.*

[6] *M. Lageman. Solaris Containers — What TheyAre and How to Use Them.http: //www.sun.com/blueprints/0505/819-2679.pdf, 2005.*

[7] *B. W. Lampson. A Note on the ConfinementProblem. Communications of the ACM, 16(10):613–615, 1973.*

[8] *J. N. Matthews, W. Hu, M.Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens. Quantifying the Performance Isolation Properties of Virtualization Systems.In Proceedings of the 2007 Workshop on Experimental Computer Science, 2007.*

[9] *National Computer Security Center. A Guide to Understanding Covert Channel Analysis ofTrusted Systems. Technical Report NCSC-TG-030,1993.*

[10] *D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. In Proceedings of the Cryptographer's Track at theRSA Conference 2006, volume 3860 of Lecture Notes in Computer Science, pages 1–20,2006.*

[11] *T. Ristenpart, E. Tromer, H. Shacham, and S.Savage. Hey, You,Get Off of My Cloud: Exploring InformationLeakageinThird-PartyComputeClouds. In Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS 2009), 2009.*

[12] *RSASecurityInc.RSADataLossPrevention(DLP) Suite.http://www.rsa.com/node.aspx?id=3426.*

[13] *R.Sailer,T.Jaeger,E.Valdez,R.Cáceres,R.Perez,*

S. Berger, J. L. Griffin, and L. van Doorn. Building a MAC-based Security Architecture for the Xen Opensource Hypervisor. In Proceedings of the 21st Annual Computer Security Applications Conference, pages 276–285, 2005.

[14] *S.Soltesz,H.Pötzl,M.E.Fiuczynski,A.Bavier,and L. Peterson. Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, pages 275–287, 2007.*

[15] *Symantec Corporation. Symantec Data Loss Prevention. http://www.symantec.com/business/data-loss-prevention.*

[16] *TrendMicro. LeakProof.http://us.trendmicro.com/ us/products/enterprise/leakproof/.*

[17] *VMware. VMware vShield Zones.http: //www.vmware.com/jp/products/vshield-zones/.*

[18] *Z. Wang and R. Lee. Covert and Side Channels dueto Processor Architecture. In Proceedings of the 22nd Annual Computer Security Applications Conference, pages 473–482,2006.*

[19] *Keisuke Okamura, Yoshihiro Oyama, Load-based Covert Channels between Xen Virtual Machines, SAC'10 March 22-26, 2010, Sierre, Switzerland. Copyright 2010 ACM 978-1-60558-638-0/10/03pg no:173-180*