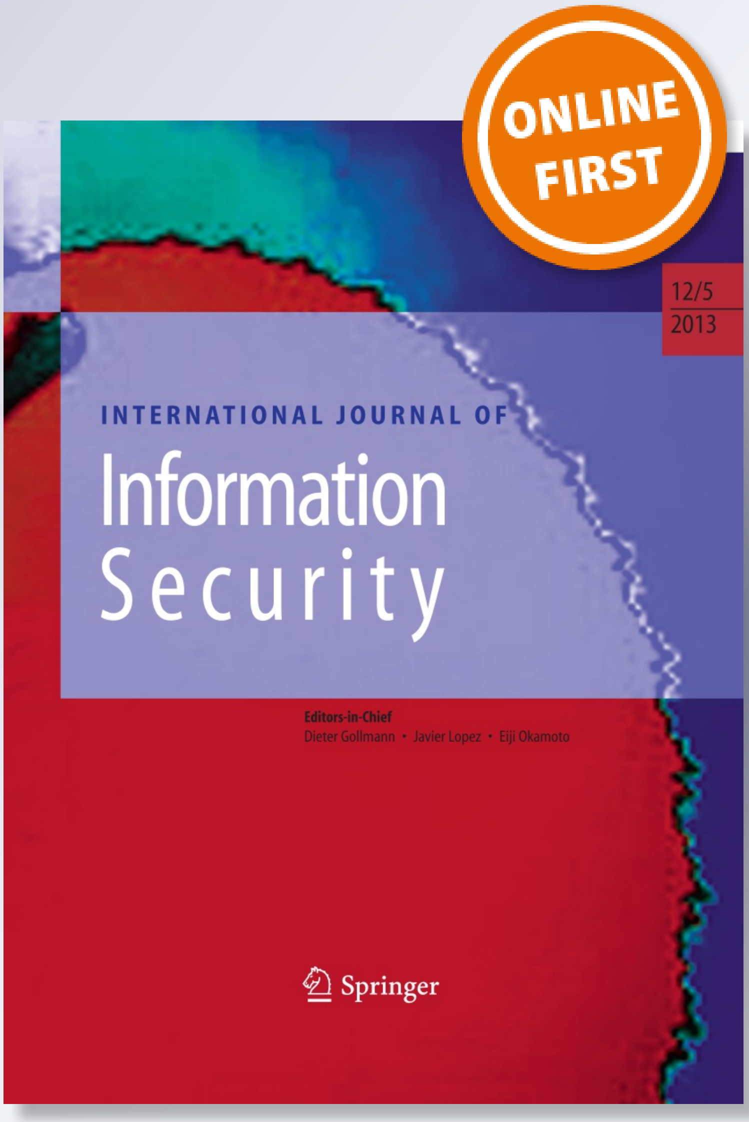
On detecting co-resident cloud instances using network flow watermarking techniques

Adam Bates, Benjamin Mood, Joe Pletcher, Hannah Pruse, Masoud Valafar & Kevin Butler

International Journal of Information Security

ISSN 1615-5262

Int. J. Inf. Secur.
DOI 10.1007/s10207-013-0210-0



Int. J. Inf. Secur.
DOI 10.1007/s10207-013-0210-0

SPECIAL ISSUE PAPER

On detecting co-resident cloud instances using network flow watermarking techniques

Adam Bates · Benjamin Mood · Joe Pletcher · Hannah Pruse · Masoud Valafar · Kevin Butler

Abstract Virtualization is the cornerstone of the developing third-party compute industry, allowing cloud providers to instantiate multiple virtual machines (VMs) on a single set of physical resources. Customers utilize cloud resources alongside unknown and untrusted parties, creating the *co-resident threat*—unless perfect isolation is provided by the virtual hypervisor, there exists the possibility for unauthorized access to sensitive customer information through the exploitation of covert side channels. This paper presents *co-resident watermarking*, a traffic analysis attack that allows a malicious co-resident VM to inject a watermark signature into the network flow of a target instance. This watermark can be used to exfiltrate and broadcast co-residency data from the physical machine, compromising isolation without reliance on internal side channels. As a result, our approach is difficult to defend against without costly underutilization of the physical machine. We evaluate co-resident watermarkingunder a large variety of conditions, system loads and hardware configurations, from a local laboratory environment to production cloud environments (Futuregrid and the University of Oregon's ACISS). We demonstrate the ability to initiate a covert channel of 4 bits per second, and we can confirm co-residency with a target VM instance in <10 s. We also show that passive load measurement of the target and subsequent behavior profiling is possible with this attack. We go on to consider the detectability of co-resident watermarking, extending our scheme to create a subtler watermarking attack by imitating legitimate cloud customer behavior. Our investigation demonstrates the need for the careful design of hardware to be used in the cloud.

A. Bates (✉)· B. Mood · J. Pletcher · H. Pruse · M. Valafar · K. Butler
Department of Computer and Information Science, University of Oregon, Eugene, OR 97405, USA
e-mail: amb@cs.uoregon.edu

B. Mood
e-mail: bmood@cs.uoregon.edu

J. Pletcher
e-mail: pletcher@cs.uoregon.edu

H. Pruse
e-mail: hpruse@cs.uoregon.edu

M. Valafar
e-mail: masoud@cs.uoregon.edu

K. Butler
e-mail: butler@cs.uoregon.edu

Keywords Cloud security · Traffic analysis · Covert channel

1 Introduction

Cloud computing has paved the way for "the long-held dream of computing as a utility" [3]. Commercial third-party clouds allow businesses to avoid over provisioning their own resources and to pay for the precise amount of computing that they require. Virtualization is a key to this model. By placing many virtual hosts on a single physical machine, cloud providers are able to profitably leverage economies of scale and statistical multiplexing of computing resources. While many models of cloud computing exist, the *Infrastructure-as-a-Service* (IaaS) model used by providers such as Amazon's Elastic Compute Cloud (EC2) service offers a set of virtualized hardware configurations for customers [2].

The sharing of a common physical platform among multiple virtual hosts, however, introduces new challenges to security, as a customer's virtual machine (VM) may be co-located with unknown and untrusted parties. Placement on a common platform entails the sharing of physical resources and leaves sensitive data processed in a cloud potentially vulnerable to the actions of malicious *co-residents* sharing the physical machine. Researchers have already demonstrated the methods of bypassing co-resident isolation in virtualization middleware, particularly through cache-based side channels [43,54,57]. Their results confirm that hypervisors present a new attack surface through which privacy and isolation guarantees can be compromised. However, defenses against such vulnerabilities are already being proposed in the academic literature [25,41].

In this paper, we consider co-residency determination alternatives that may be available even if current avenues for exploitation no longer exist. We focus on investigating the network interface, a channel that is explicitly communicative and is a multiplexed resource in virtualized settings. We use concepts explored in the area of active traffic analysis to develop an attack that uses a physical machine's network interface to create an outbound covert channel for data exfiltration. Our attack can be carried out with a malicious CLIENT contacting a victim machine in the cloud (e.g., a Web server or media server, hereto referred to as the SERVER) and observing the throughput of traffic received. In collaboration with a FLOODER deployed in the cloud, we examine interpacket delays and the corresponding distribution of packet delays from the server to determine whether the FLOODER has become co-resident with the SERVER, using a Kolmogorov–Smirnov distribution test to make this determination. In general, there is limited visibility into the cloud, but we correlate ground-truth measurements based on out-of-band communication with production cloud providers to validate our results. We show that despite different network packet scheduling strategies among hypervisors used in clouds, our attack is implementation-independent. We can determine whether instances are co-resident in under 10 s and in as few as 2.5 s for a given probe. We further describe how a covert channel can be deployed that can transmit 4 bits per second, and describe how our attack can be used to perform passive load measurement on the victim SERVER, allowing us to profile its activity.

This paper makes the following contributions:

– *Investigates virtualization side channels in physical hardware* Previous research in cloud security has investigated sharing at the hypervisor software layer. Our work takes a bottom–up approach by considering whether or not hardware designed for non-virtual environments is safe for cloud deployment. We make the surprising discovery that technologies designed to aid virtualization such as SR-IOV and VMDq actually facilitate co-resident watermarking.
– *Assesses severity of threat through extensive evaluation* We determine the practicality of our attack through an extensive series of tests. These tests demonstrate co-resident watermarking's robustness under Xen, VMWare ESXi, and KVM hypervisors, with varying server loads, network conditions, and hardware configurations, and in geographically disparate locations. In a final test, we employ our scheme in a production science cloud to successfully watermark a target network flow within 2.5 s.
– *Introduces proof-of-concept attacks for the network flow channel* We develop an accurate load measurement attack that explicitly detects and filters out the activity of other virtual machines, an issue left unaddressed in the previous work [43]. We also demonstrate the creation of a covert channel capable of transmitting 4 bps of information.
– *Develops detection-avoidance strategies for cloud watermarking* The inherent noise of compute cloud data centers offers advantages in the development of detection-avoiding network flow watermarks. We enhance our original scheme by masking the delay signal in innocuous cloud customer activity and discuss how this scheme could be further adapted to behave as specific cloud-based public Web services. Using an extremely conservative parameterization, our proof-of-concept implementation can confirm co-residency with minimized risk of detection in under 2 min.

The rest of this paper is organized as follows. We provide a brief introduction to the issue of cloud co-residency in Sect. 2 and present the relevant concepts of active traffic analysis, particularly network flow watermarking, in Sect. 3. Section 4 presents a threat model and our co-resident watermarkingencoding and decoding steps. In Sect. 5, we elaborate on the application of our scheme. Our attack is thoroughly evaluated in Sect. 6, where it is tested under various conditions. Practical use scenarios are considered in Sect. 7. We adapt our original attack to frustrate detection attempts in Sect. 8 and discuss countermeasures to the co-resident network flow side channel in Sect. 9. Related work is considered in Sect. 10 before we conclude in Sect. 11. An overview of hypervisor resource-sharing mechanisms is included in Appendix A, as well as an introduction to virtualization-aware hardware in Appendix B.

2 Cloud co-residency

In compute clouds, the co-resident threat considers a malicious and motivated adversary that is not affiliated with the cloud provider. Victims are legitimate cloud customers that are launching Internet-facing instances of virtual servers to do work for their business. The adversary, who is perhaps a business competitor, wishes to use the novel abilities granted to him by cloud co-residency to discover valuable information about his target's business. This may include reading private data or compromising a victim machine. It could also include more subtle attacks such as performing load measurements on the victim's server or launching a denial-of-service attack. Masquerading as another legitimate cloud customer, the adversary is free to launch and control an arbitrary number of cloud instances. As is necessary for the general use of any third-party cloud, the cloud infrastructure is a trusted component.

Co-residency detection through virtualization side channels is a danger that was first exposed by Ristenpart et al. [43]. This work lays out strategies for exploiting the instance placement routines of the Amazon EC2 cloud infrastructure in order to probabilistically achieve co-location with a target instance. From there, co-residency can be detected using a cross-VM covert channel as a ground truth. While more advanced methods of successful placement are outlined, such as abusing temporal locality of instance launching, it is shown that a brute force approach is also modestly successful. Masquerading as a legitimate customer, an attacker is able to launch many instances, perform the co-residency check, terminate, and repeat until the desired placement is obtained. Several cross-VM information leakage attacks are also outlined, such as the load profiling and keystroke timing attacks.

However, we independently confirmed that many of the approaches in previous work, such as the use of naive network probes, are no longer applicable on the EC2. This, combined with academic proposals that better isolate cross-VM interference impacts [41], makes co-residency detection significantly more difficult at this time. Instead, we introduce an alternate viable co-location test, co-resident watermarking. In our exploration of potential defenses, we conclude that closing the employed covert channel is difficult without costly dedicated hardware or reduced network performance. Our approach is not dependent on adversarial advantages such as cloud cartography and placement locality that were available in [43], although these would still ease the work of the attacker.

3 Active traffic analysis

Our approach uses concepts previously explored in network flow watermarking and other active traffic analysis attacks. Network flow watermarking is a type of network covert timing channel [11,12], capable of breaking anonymity by tracing the path of a network flow. Normally requiring the cooperation of large autonomous systems or compromised routers in anonymity networks, a target's traffic is subjected to controlled and intentional packet delay at an institutional boundary in order to give it a distinct and recognizable pattern [23,24,50,56]. When the traffic exits the institutional boundary, that pattern is still present and can be decoded. Network flow watermarking can be employed to perform a variety of traffic analysis tasks. They are of great interest recently as a method for detecting *stepping stone* relays [7,14,33,51], and compromising network anonymity services (e.g., TOR network) [27,34].

Previous work has considered a number of challenges in the design of a watermarking scheme. Schemes can be grouped into blind and non-blind approaches. In blind schemes, the watermarking parties do not store any state information for their target. All of the necessary information is contained within the watermark, which is itself a side channel. In a non-blind scheme, state information about the target is stored for access by the exit gateways. Watermarks must be *robust* to modifications from network traffic and jitter. If the watermark is also resistant to intentional tampering or removal, it is said to be *actively robust*. Watermarks are also ideally *invisible* so a target cannot test for its presence. If detection mechanisms such as the multi-flow attack are viable, the target can recover the secret parameters and remove the watermark [27]. However, recent work has shown that even the most advanced schemes do not possess the invisibility property [11,20,34]. As such, the preliminary focus of this work is to determine the efficacy and throughput of the co-resident network flow channel. After quantifying the effectiveness of our approach, we then leverage the unique advantages of the compute cloud domain to develop an invisible watermark scheme in Sect. 8.

Our methodology also bears similarities to previous work on traffic analysis of Tor. Murdoch et al. [36] demonstrated that a corrupt Tor node can collude with a network server to extract information about the path of a Tor connection. This is accomplished through latency measurements of Tor relays after filling the connection with probe traffic. These results were novel and troubling in that Tor necessarily relied on mixing traffic from different sources to establish anonymity. Our work exploits virtualization's dependence on traffic mixing to improve performance and resource utilization. Critically, our work differs from [36] in that it does not require a corrupt network server. Instead, we rely on a colluding VM to manipulate the behavior of its co-resident victim.

4 System design

We next present a simple scheme that can be applied from the co-resident position to inject a target's network traffic with a persistent watermark. Given a sufficiently long network flow, it can break hypervisor isolation guarantees regardless of cloud or network conditions. Due to the coarse-grained abilities of a co-located VM to inject network delay, we employ an ON- OFF interval-based packet arrival scheme rather than attempting to control the delay between individual packets. Our scheme leverages out-of-band communication between the encoding and decoding points in order to overcome its limited ability to inject delay through network activity.

4.1 Threat model

This work's primary motivation is to investigate the existence of hardware-level side channels in cloud infrastructures, calling into question the viability of isolation assurances for virtual machines. We go beyond the traditional co-resident threat model and imagine a cloud in which naive timing channels such as network probes are unavailable to the adversary; cloud administrators have chosen to route all local traffic through a switch that fuzzes the results of these services and prevents co-residency detection. To their credit, the administrators in this cloud have also proactively applied patches that have all but eliminated popular cache-based hypervisor side channels. Given the relatively small attack surface that the virtual hypervisor represents, this is not too imaginative of a leap. In fact, we observed that some of these security measures had been taken in our own investigation of EC2. In spite of these obstacles, our adversary wishes to discretely discover his victim in the cloud through innocuous use of his own instances.

We assume system administrators are not interfering with the activities of their customers and will not intervene with customer behavior unless it is a threat to service level agreements (SLAs) or to the general health of their business [1]. We also assume that our victim is trusting the cloud infrastructure and expects modest delays imposed by other cloud customers. From the isolation of their VMs, the victim will be unable to make inferences about the cause of variances in system performance. As a result, the victim is unable to differentiate between the activities of the adversary and the actions of other legitimate cloud guests. Finally, we assume that the victim's instances are available to the adversary over an open network and that the adversary is able to create network flows from these instances on the order of several seconds.

4.2 Co-resident watermarking

Like previous work in cloud co-residency, the co-resident watermarking attack relies on the pigeonhole principle to probabilistically achieve co-location with a victim virtual machine, launching many virtual machines and then performing statistical side channel tests from each [43]. To begin the search for his target, the attacker launches a large number of instances on the cloud. We refer to these instances as FLOODERs. Each FLOODER announces its presence to a master host, the CLIENT, which is a colluding agent situated outside of the cloud. The attack begins when the CLIENT initiates a Web session with our target instance, the SERVER, which is accessible at a predetermined IP address. Systematically, the CLIENT iterates through its list of registered FLOODERs, sending a series of signals to each. Based on these signals, the FLOODER injects network activity into the outbound interface of its physical host machine. This activity is multiplexed

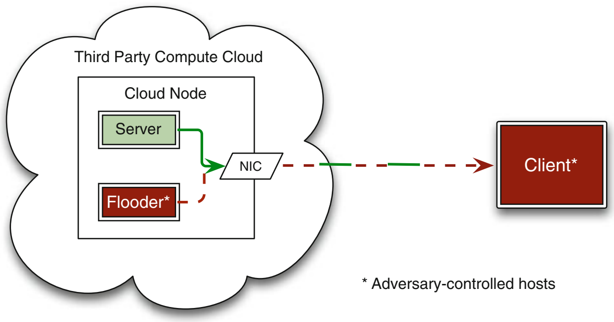


Fig. 1 The attack model considered for co-resident watermarking. Two colluding hosts, the CLIENTand FLOODER, attempt communicate through the legitimate network flow of the SERVER

with the outbound traffic of the server, creating delay in the legitimate SERVER flow. This delay constitutes the building block of our watermark scheme. In the event that a FLOODER is co-resident to the SERVER, the CLIENT–SERVER flow can be imprinted with a watermark signature. This creates a beacon through which the CLIENT can test for co-location. The CLIENT tests each FLOODER's location for a portion of its network flow. If no watermark signature is detected, the attacker can terminate all instances and launch a new set until co-location is achieved. In the event that a signature is detected, the attacker can use the co-resident FLOODER for a second phase of attack. This could involve another known exploit or continued use of the network flow side channel. Our co-resident watermarking attack is pictured in Fig. 1.

4.3 Signal encoding

In this section, we explain the watermark embedding process. An unwatermarked network flow of length T between a cloud server instance and a remote client can be divided into n intervals of length t_i. Each interval t_i will observe a certain number of packet arrivals p_i over its portion of the network flow. Traditionally, the encoding of a watermark requires that two different levels of packet delay, d_+ and d_-, be repeatedly and randomly introduced to a network flow with equal probability. These two delay levels form the bits to be read from the side channel. The watermark is therefore made up of components $\{w_i\}_{i=1}^{n}$ where

$$w_i = \begin{cases} d_+ & \text{with probability } \frac{1}{2} \\ d_- & \text{with probability } \frac{1}{2} \end{cases}$$

From the co-resident position, we are limited in our ability to inject arbitrary amounts of delay into the flow, nor can we inject a negative amount of delay. Therefore, our delay values (d_+, d_i) represent the maximum and minimum total amount of network activity we are able to introduce from a co-resident virtual machine. Upon receiving a signal to mark the flow, d_+ is achieved through a co-resident FLOODER host injecting a constant stream of UDP packets onto the network interface. Conversely, d_- is achieved through taking no action for the length of the interval.

In addition to the activities of co-resident instances, the variance in p_i will reflect other system noise: hypervisor scheduling, network congestion, and virtualization-imposed artifacts. While these factors will not remain constant for any meaningful length of time [44], their effects can be ignored due to the random assignment of delay value d_+ or d_- to each sequential w_i. The statistical measure for decoding watermark signals described in Sect. 4.4 filters this noise; over a sufficiently long trial, system noise impacts both delay values (distributions) equally. In Sect. 6, we demonstrate that watermark signals can be decoded in spite of the presence of any of these factors.

4.4 Signal decoding

At the decoding point, packet arrivals per interval are recorded over the length of the flow. After each measurement, the intervals are sorted into samples X_{d_+} and X_{d_-} based on the prenegotiated co-resident activity representing d_+ and d_-, and with X_{d_+} and X_{d_-}, respectively, containing n_+ and n_- measurements. If co-residency has been achieved, then these two interval groupings represent the flow during two distinct network states. We can therefore expect the interval grouping samples to have different discrete distributions.

These two samples can be compared using statistical similarity tests. In this work, we employed the nonparametric Kolmogorov–Smirnov (KS) test for independence [40]. This statistical measure has been employed previously in other analysis of covert timing channels [20,38]. To test the null hypothesis that the two samples are from the same distribution, a statistic is calculated and compared to a look-up value corresponding to 95 % confidence. If the test fails, then the decoder rejects the similarity of the distributions and declares the instances to be co-residents.

For the Kolmogorov–Smirnov test, the decoder calculates the empirical cumulative distributions $F_{1,n_+}(X_{d_+})$ and $F_{1,n_-}(X_{d_-})$. The KS statistic is then calculated as follows:

$$D_{n_+,n_-} = \sup|F_{1,n_+}(X_+) - F_{1,n_-}(X_-)|,$$

where sup is the supremum of the differences in the cumulative distributions. The null hypothesis can be rejected with confidence α if

$$\sqrt{\frac{n_+ n_-}{n_+ + n_-}} D_{n_+,n_-} > K_\alpha,$$

where K_α is a critical value from the Kolmogorov distribution.

Both sides of this relation can range in value from 0 to 1. The three factors to consider in the relation are the α value, the KS statistic representing the greatest difference between the two distributions (D_{n_+,n_-}), and the sample sizes. Smaller α values correspond to greater confidence levels and thus increase the value of K_α (e.g., for a given n, $K_{.05} > K_{.1}$). The significance of the KS statistic increases with either the size of the supremum or the sample size. Hence, the KS test may succeed with a large supremum over a short number of measurements or a small supremum over a large number of measurements.

An alternate nonparametric test that is better known for use with discrete distributions is the Pearson's chi-square (χ^2) test. We chose not to use this metric because of the difficulty of handling the trivial case in which samples are extremely dissimilar. χ^2 struggles with any cell frequencies that are less than 5, and quite often in our evaluation, we found that the FLOODER's impact was such that there was no overlap in the contingency tables of the marked and clear intervals. Relying on the χ^2 test would have also hindered our ability to make swift determinations of co-residency.

5 Implementation

Our target instance, the SERVER, was a virtual machine serving static content via Apache 2. The CLIENT host initiated a TCP session with the SERVER, continuously re-requesting a 10 MB file. To create more realistic Web traffic conditions, we wrote a PHP script that simulated background noise on a server. The script conservatively estimated a 1:3 write-to-read traffic ratio, creating a more turbulent channel from which to perform our measurements. Upon execution, the script reads a bounded amount of non-cached data from a file, optionally executes a disk write, and finally performs a CPU-bound set of computations. This closely models applications on a production Web server, where for each request, the server will fetch data from a database, perform some computation or transformation on it, and return it to the user. Alternately, in the case of the disk write, this models the other common case seen inside Web applications where a user sends data, computation is performed, and the data are written to disk. As read requests are more common for many Web servers, we weighted these probabilities accordingly. To simulate the activity of additional cloud customers, a GUEST VM ran a script that behaved similarly to the SERVER.

Our FLOODER used a raw socket injection binary, written in C, that responded to prompts from a CLIENT host to create outbound multi-threaded UDP streams for specified intervals. The packet streams were directed by MAC address to a neighboring cloud instance that was not otherwise a participant in the trial. Alternately, the FLOODER could set the time-to-live of packets to 0 and direct the flood to a host outside of the cloud. The former is a more appealing option, as it decreases the cost of the attack on services, such as Amazon EC2, that have fees for data transferred into and out of the cloud. Under either design, the FLOODER's activity passes through the network interface and then immediately leaves the path of the CLIENT–SERVER flow. In Sect. 6.6, we demonstrate that this is sufficient to avoid secondary bottlenecks that might lead to false positives in our co-residency check.

The CLIENT monitored the watermark impact by signaling the FLOODER and performing synchronized reads on the network flow between the CLIENT and SERVER. The flow was measured by monitoring the number of packet arrivals by interval. Synchronization was established through estimating the round-trip time between the CLIENT and FLOODER. Various hypervisors introduce additional delays and artifacts through their fair resource scheduling algorithms. In order to ensure the FLOODER's effect was captured, we limited the hypervisor's ability to react to the FLOODER's activity. We measured in small bursts of 250 ms and then waited 2 s before signaling the flooder again. This was sufficient to ensure that our measurements were independent. The Kolmogorov–Smirnov test was then applied using the scipy statistical library.

6 Evaluation

We used a number of different testbeds to evaluate our approach, as shown in Fig. 2. The first was a local area network that contained a commodity switch, two Dell workstations and one Dell PowerEdge R610 server with two 4-core Intel Xeon E5606 processors and 12 GB RAM. Each machine had a network interface card that could transmit in 1000 BaseT. In a subsequent trial, we replaced the server's NIC with an SR-IOV enabled Intel 82599 10 Gbps Ethernet controller and attached it to the LAN with a fiber-to-copper Ethernet transceiver. The server was dual-booted with both VMWare ESXi 4.1 and a Xenified Linux 2.6.40 kernel. On both hypervisors, we launched two or more similarly provisioned virtual machine guest images that acted as our cloud instances. Each VM ran the Linux 2.6.34 kernel allocated with resources similar to those afforded to an Amazon EC2 small instance, approximately 1 vCPU compute unit and 1.7 GB memory.

Additionally, we used two science clouds for further analysis. The first, the University of Oregon's ACISS, ran OpenStack KVM. Here, each guest image was provisioned with 1 vCPU and 2 GB memory. The instances received network access through a bridged 10 GbE network card. Each physical host was connected to a 1:1 provisioned Voltaire 8700 switch with fiber channel. The switch had 2 10 GbE trunks to a Cisco router that connected to the university network. The second cloud was Futuregrid's Sierra at the San Diego Super Computer Center. Sierra ran the Nimbus service package with the Xen 3.0 hypervisor. Instances on Sierra were also bridged onto a 10 GbE switch. Each of the investigated hypervisors used default network management configurations, and none imposed traffic shaping or bandwidth ceilings on the managed VMs.

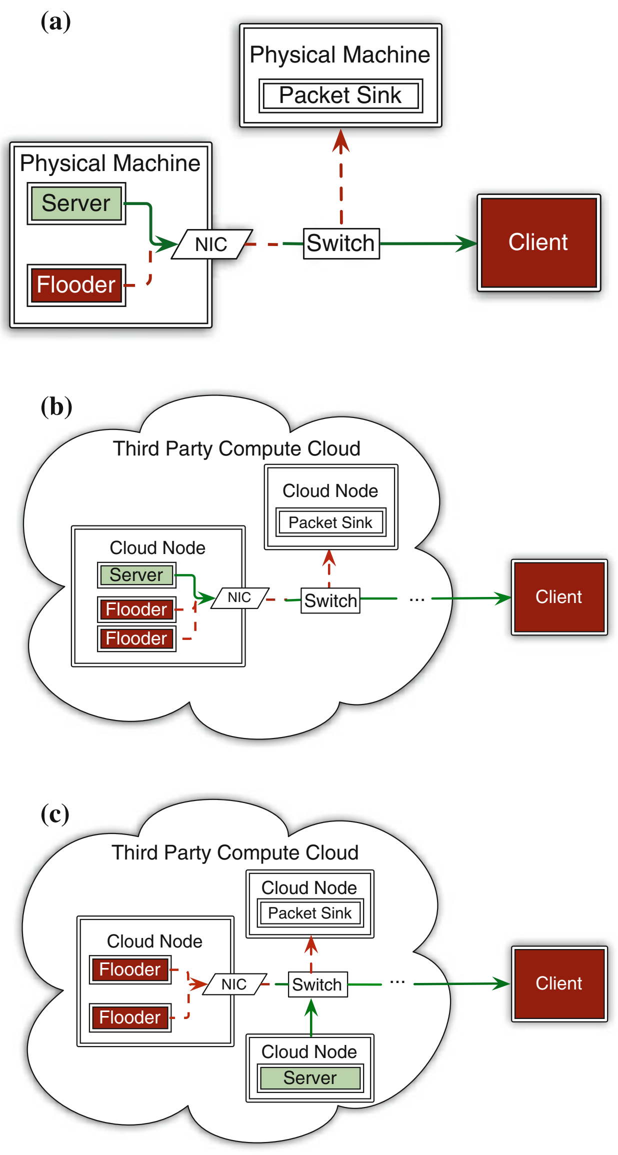


Fig. 2 Testbed topologies used in evaluation. Adversary-controlled hosts are shaded in *red*. **a** Local testbed, **b** cloud, successful co-location, **c** cloud, failed co-location (color figure online)

The CLIENT process requires little processing power and can be run from any commodity PC or reasonably provisioned virtual machine. On our local testbed, it was run primarily from a bare metal workstation running a Linux 2.6.40 kernel with 4 GB memory and a Pentium dual-core 3 GHz CPU. The workstation had a NIC that was supported to 1000BaseT full duplex. We used additional performance tools to confirm that the CLIENT host was sufficiently provisioned to handle these tasks. To test our ability to decode the watermark with longer paths and realistic network conditions, we launched CLIENT instances that performed the watermark attack on our local testbed from a bare metal machine at a geographically disparate university. This instance was running a Linux 2.6.38 kernel with 8 GB memory, an Intel Xeon X3450 2.67 GHz processor, and a NIC set to 1000BaseT full duplex.

6.1 Xen hypervisor

We first attempted our co-resident watermarking scheme using the local Xen testbed. This configuration is pictured in Fig. 2a. The default Xen bridged networking settings were used for domU's virtual interfaces, which were set to 100BaseT full duplex. As we note in Appendix A.1, Xen's dom0 bridge imposes major delays and represents the transmission bottleneck of this first test. Although this does not exploit the physical interface, we chose to examine this Xen configuration due to its popularity. Subsequent trials demonstrate that our approach is not dependent on any particular hypervisor or network interface.

For this initial test, the CLIENTinitiated a single TCP session with the SERVER's `apache` process. The CLIENTthen generated a random binary signal that was transmitted to a FLOODER, causing it to generate intermittent UDP traffic floods. The CLIENT measured packet arrivals by interval and sorted these into marked (d_+) and clear (d_-) samples. The probability density of these two samples is pictured in Fig. 3a. This figure and all others are based on 3,200 total measurements that correspond to 13 min and 20 s of observed network flow. Immediately after the trial, a second control test was launched in which the FLOODER was not signaled and took no action.

Based on visual inspection alone, it can be observed that there is great similarity between the packet arrival distributions for the clear intervals and the undisturbed control flow. In contrast, there is great difference between the distributions of the clear intervals and marked intervals. After just 2.5 s of observed network flow, the KS statistic for the clear and marked distributions is 0.98. The p value, which represents the likelihood of obtaining such an extreme result under the null hypothesis, is 0.01. This is sufficient to reject the null hypothesis, and confidence only increases throughout the remainder of the trial. In contrast, comparing the clear interval sample to the control flow yields a KS test statistic of 0.38, which is insufficient to reject the null hypothesis at 95 % confidence. This is because $K_\alpha = .41$ with $n = 10$ and $\alpha = 0.05$.

The combination of these two tests is sufficient to declare that our instances are co-located. For the remainder of our evaluation, we will discuss only the KS test in which the clear and marked intervals are compared. In each trial, though, we also apply the KS test to the clear and control samples in order to confirm that they are statistically indistinguishable.

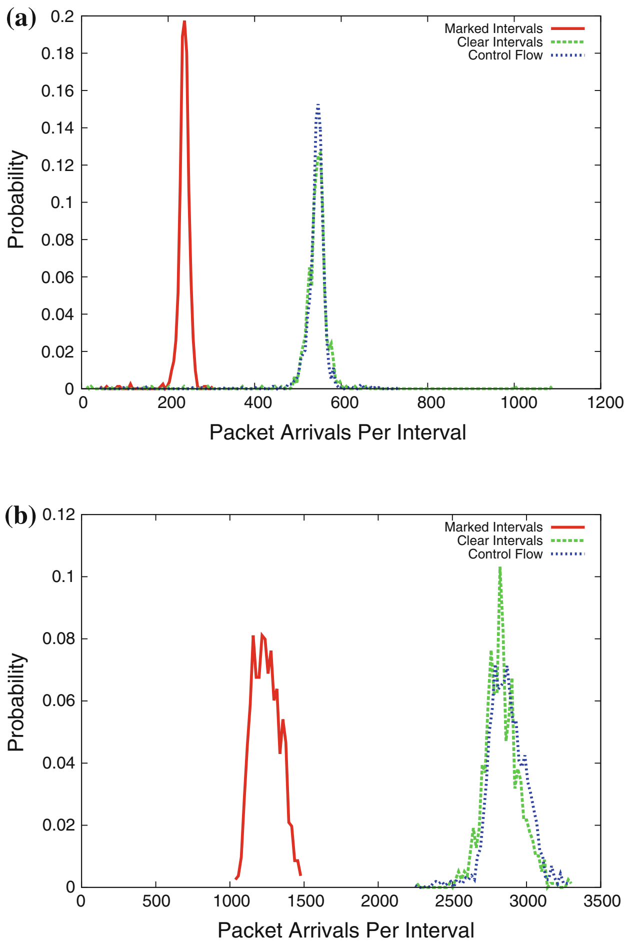


Fig. 3 Probability density functions for co-resident watermarking. *Marked intervals* represent the distribution for delay value d_+. *Clear intervals* represent the distribution for delay value d_-. *Control flow* represents the distribution for a second trial in which the FLOODER VM takes no action, **a** Xen on local testbed, **b** VMWare ESXi on local testbed

6.2 VMWare ESXi hypervisor

To determine whether differences in hypervisor scheduling affect our watermarking results, we repeated the above trial on the same testbed, now using the VMWare ESXi hypervisor. In contrast to Xen, ESXi does not have an administrative domain through which packets are routed. Instead, packet transmission requests are sent directly to the hypervisor and are scheduled with a simple round-robin algorithm (see Appendix A.2). Resultantly, ESXi is much more efficient at packet transmission. The results, shown in Fig. 3b, show that that our SERVER running on ESXi enjoys significantly higher throughput than Xen under similar conditions. Once again, the unmarked sample is similar to the control flow, but dissimilar to the marked sample. As there is no overlap between the clear and marked intervals, the KS statistic is 1. We are once again able to reject the null hypothesis, confirming that our FLOODER is co-resident to the SERVER. This

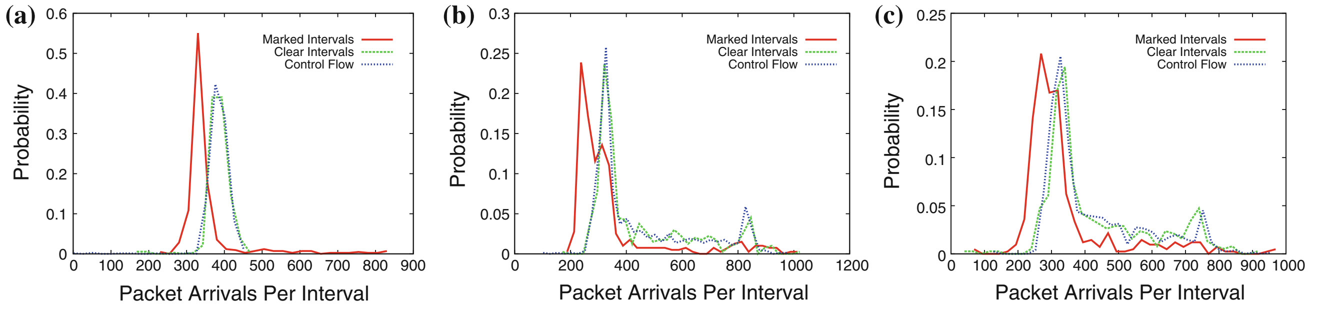


Fig. 4 Probability density functions for co-resident watermarkingwith increasing numbers of I/O bound Web server guest instances. These trials were otherwise identical to those run in Sect. 6.1. **a** 1 additional GUEST, **b** 2 additional GUESTs, **c** 3 additional GUESTs

Table 1 Results of tests in Xen as system load increases

Trial	Length (s)	KS_{d_+,d_-}	p value	Result
SERVERand FLOODER	2.5	0.99	0.01	*Co-Res*
Add 1 GUEST	3.75	0.78	0.05	*Co-Res*
Add 2 GUESTs	3.75	0.91	0.01	*Co-Res*
Add 3 GUESTs	10	0.49	0.05	*Co-Res*

Length is the minimum flow lengths required to achieve 95 % confidence. KS_{d_+,d_-} is the KS statistic that compares the clear and marked samples after *length* seconds. p value is the likelihood of obtaining such an extreme result under the null hypothesis

demonstrates the feasibility of co-resident watermarkingon two of the major hypervisor families.

6.3 System load

To demonstrate its applicability in real cloud environments, we assessed the ability of the FLOODER to inject a watermark signature under increasingly adverse system conditions. In addition to launching FLOODER and SERVER instances on our local Xen testbed, we launched an increasing number of GUEST instances. These GUESTs represent other communication-intensive customers in the cloud that are non-participants in our attack. Each GUEST behaved identically to the SERVER, running Apache and serving up files over prolonged HTTP sessions.

We repeated our standard trial with up to 3 GUESTs for a total of 5 instances on the machine. This load approached the maximum capacity of our testbed. The results of these trials are pictured in Fig. 4a–c. As the number of GUESTs on the machine increases, we see distribution of the marked samples begins to approximate the distribution of the clear samples. From this, we suspect that extreme load can potentially erase our watermark signature. However, the Kolmogorov–Smirnov test offers a more precise measurement than visual observation. These results, shown in Table 1, demonstrate that co-resident watermarkingcan be used to confirm co-residency in 10 s with 95 % confidence, even when system load is high.

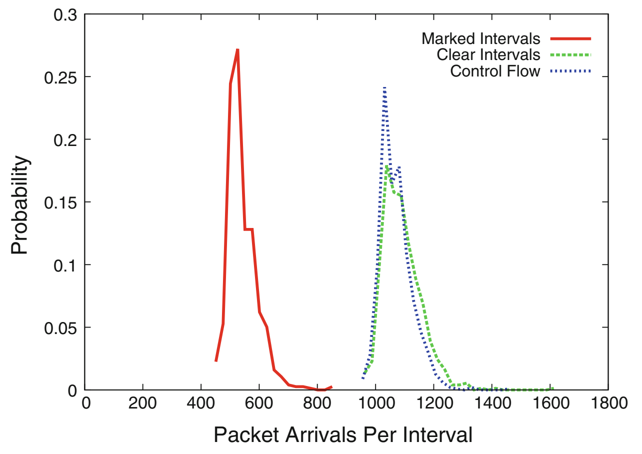


Fig. 5 Probability density functions for co-resident watermarkingon Xen over long network path

6.4 Network conditions

Our next experiment measured the resiliency of encoded watermarks when traveling across longer network paths. To do this, we executed our CLIENT process from a bare metal host at a geographically disparate university. The CLIENT issued HTTP requests to the SERVER that resided on our local Xen testbed. To smooth the observable network flow in the presence of higher round-trip times, the CLIENT initiated 5 TCP sessions with the SERVER. Results from this long-distance trial are pictured in Fig. 5. Once again, there is no visible similarity between the clear and marked distributions. The watermark signature is still identifiable after just 2.5 s and yields a KS statistic of 1 (p value 0.01). We are once again able to reject the null hypothesis, confirming that our FLOODER is co-resident to the SERVER. The persistent presence of the watermark means that the co-resident watermarking attack is not distance bounded relative to the location of the cloud provider.

6.5 Science clouds

Having found success on our local area network, we set out to replicate our results on industry-class hardware in a partially controlled environment. We used the ACISS compute cloud service as well as Futuregrid's Sierra cloud at the San Diego Supercomputing Center. On the private science cloud, we were able to launch two instances that were confirmed to be co-resident by the cloud staff. On Sierra, we confirmed co-residency by querying the Nimbus cloud client for the physical host of our instances. We did not have any foreknowledge of the activity of other users in these clouds. Our initial attempts to launch co-resident watermarking in this environment failed due to an unforeseen network bottleneck. On ACISS, we were only able to generate approximately 3.2 Gbps of traffic from a single FLOODER instance, falling well short of the 10 Gbps channel. Through consultation with the cloud staff, we discovered that this bottleneck was due to the virtio_net driver, which has been shown by the KVM community to be incapable of filling a 10 Gbps link with a single VM [55]. We encountered similar problems with Xen in the Sierra deployment, which used a paravirtualized network configuration that imposed even greater delays on packet transmission (see Appendix A). Thus, we were prevented from injecting packet delay into the CLIENT–SERVER flow. Because we were only off by a small constant factor, though, we reattempted the trial with multiple co-resident FLOODERs. This topology is pictured in Fig. 2b. While achieving "tri-residency" would not be a realistic attack scenario, this served as a stand-in for a more sophisticated denial-of-service attack against the physical network interface. Additionally, as many cloud applications are communication intensive [19], we can expect some of the difference in bandwidth to be made up for by the activities of other cloud customers. Recent work in VM network performance enhancement, if adopted and deployed, could also increase the instance throughput sufficiently to make tri-residents unnecessary [19,42].

The results of these trials are visible in Figs. 6 and 7. In spite of the unknown and uncontrolled state of the cloud cluster, the watermark signature between the clear and marked interval samples is still clearly visible. After 5 s of observed flow on the ACISS cloud, the result is a KS statistic of 0.98 with a p value of 0.01. We are once again able to reject the null hypothesis, confirming that our FLOODER is co-resident to the SERVER. These results demonstrate the feasibility of co-resident watermarking for the KVM hypervisor. Under similar conditions, we successfully launched co-resident watermarkingon a Futuregrid cloud. The KS test yielded a statistic of 0.97 with p value of 0.02 after 2.5 s of observed flow. These tests demonstrate that our current implementation is nearly practical for industrial compute clouds.

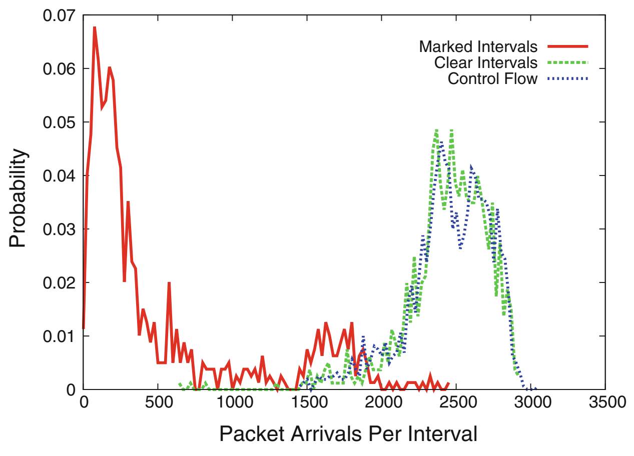


Fig. 6 Probability density functions for co-resident watermarkingon University of Oregon ACISS (KVM)

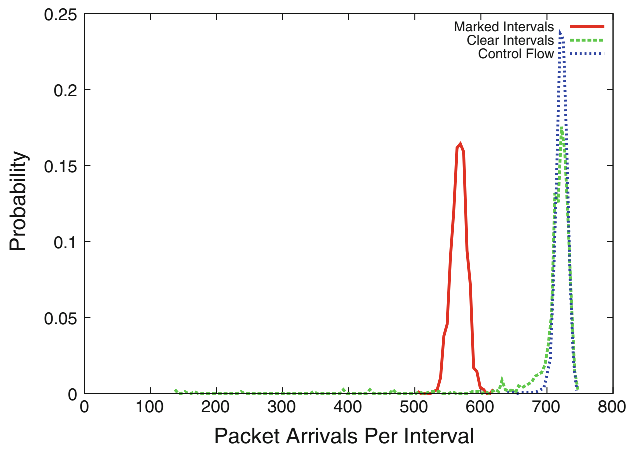


Fig. 7 Probability Density functions for co-resident watermarkingon Futuregrid Sierra (Xen)

6.6 Neighboring node false positives

We have shown co-resident watermarkingto be capable of detecting co-residency in a variety of circumstances. However, for this attack to be practical, it must also avoid false positives, reports that the FLOODER is co-located with the SERVER when it in fact is not. This is of greatest concern for topologies in which the instances are not co-resident but share a common network path. In order to be multiplexed at the network interface, the FLOODER's activity necessarily must reach the first switch; if packets are resultantly delayed at this point, then the watermark signature would be injected on all network flows that share the switch. Due to our design decision to inject layer 2 packets that are routed by MAC address to another adversary-controlled instance, we know that the FLOODER and SERVER flows' paths share only a single hop.

To confirm that co-resident watermarkingis not susceptible to false positives, we configured a new topology on

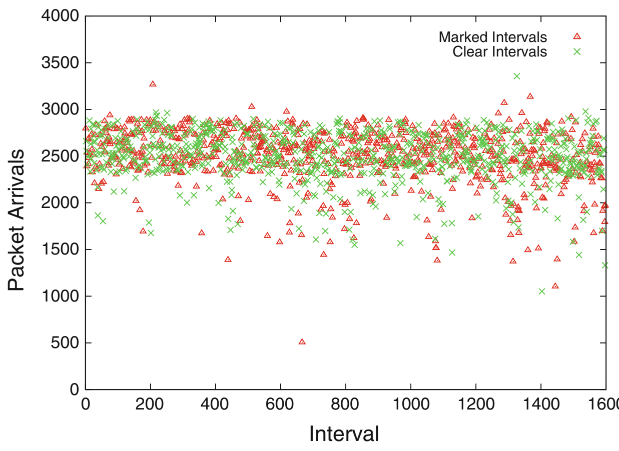


Fig. 8 Packet arrivals on ACISS (KVM) when the Server and Flooder are hosted on different nodes. The high level of overlap between marked and clear traffic indicates that flooding activity does not impact neighboring physical machines

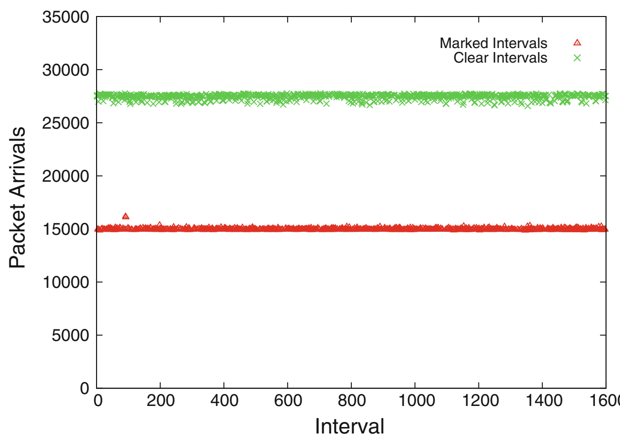


Fig. 9 Packet arrivals on local testbed using SR-IOV network interface and driver. The marked and clear traffic have no overlap. Throughput is much higher using the SR-IOV NIC; unmarked traffic transmitted data at 0.83 Gbps, with the marked traffic at 0.41 Gbps

ACISS in which the Server was *not co-resident* to the Flooders, but shared a common upstream switch one hop away. This topology is pictured in Fig. 2c. We confirmed this topology through ARP table inspection and conferring with the cloud staff. We then repeated the trial. The results are pictured in Fig. 8. The activity of the Flooders does not appear to impact neighboring instances. In fact, the clear intervals and marked intervals yield a KS statistic of 0.46 and p value of 0.65 after 2.5 s of observed network flow. The test becomes more conclusive over prolonged observation, with a statistic of 0.11 and p value of 0.37 after 5 min. The two samples are not distinct enough to reject the null hypothesis; the test concludes that they were drawn from the same distribution.

6.7 Virtualization-aware hardware

To investigate the viability of hardware-level defenses against co-resident watermarking, we repeated our original Xen trial on an SR-IOV-enabled NIC. SR-IOV [30] is a specification that allows physical I/O devices to present themselves to the host as multiple virtualized I/O devices, allowing for direct access to PCI interfaces. This especially impacts network access in Xen, eliminating the need for dom0to be involved in copying packet buffers from the guest domain. Since each domUhas access to its own PCI virtual function, SR-IOV also provides individual queues for each VM. Arriving packets are sorted into these queues based on their destination, then are copied directly to the guest OS memory using DMA. We discuss virtualization pass-through technologies further in Appendix B.

We tested our watermarking technique using an Intel 82599 ES 10 Gbps Ethernet controller that supports the SR-IOV specification using the `ixgbe` driver. We configured the driver to present two virtual functions (VFs) on a single outgoing port, which appear as separate PCI devices. We then connected the Server and Flooder to one VF each on our Xen testbed. The outbound port was connected to our local workstation with a fiber-to-copper Ethernet transceiver; this reduced the bandwidth of the Ethernet controller, but preserved the driver's behavior.

The results from this trial are shown in Fig. 9. We observe that by eliminating the middleware of the virtual hypervisor, co-resident watermarking has become even more effective. When both the Flooder and Server are actively filling their dedicated packet queues, each receives roughly 50 % of the available system throughput (∼0.41 Gbps). When the Flooder is inactive, the Server is able to transmit at the highest possible rate (∼0.83 Gbps). The KS test trivially rejects the null hypothesis. The Flooder's ability to have such an impact indicates that, unlike hypervisor-managed network sharing schemes, the ixgbe driver imposes no fairness measures based on anomalous virtual machine behavior. As a result, the bandwidth of our side channel had increased due to virtualization-optimized hardware. The security ramifications of future performance-driven enhancements for virtualization need to be carefully considered before their adoption.

7 Analysis

We have demonstrated that co-resident watermarkingis capable of bypassing VM isolation and exploiting underlying hardware configurations. There are a variety of circumstances in which an attacker could consider making use of the outbound traffic side channel. Traditional co-resident threats such as covert communication and load measurement are considered below. In a laboratory environment, we developed a proof-of-concept attack for each. In future work, we hope to further develop these approaches.

Co-resident watermarking's low cost makes it an appealing scouting mechanism to precede the use of a more devastating exploit, such as a zero day against the hypervisor. The exact cost of launching this attack depends on the cloud being considered. However, we can provide a rough estimate by using the results of Ristenpart et al.'s brute force attack, in which an 8.4 % placement was obtained over 1,684 targets using 1,784 probes. At the current minimum instance charge of $0.08 per hour for small Amazon EC2 instances, our attack would cost $1.01 and require 6 min, 20 s per successful co-location. Note that this estimate is based on the full minimum hourly charge imposed by Amazon, even though the time required by the probe is much less; the low cost per co-location is obtained through having a large set of targets. We also assume here that the CLIENT is also an EC2 instance, thus avoiding additional fees for outbound cloud traffic. While Amazon's cloud services have expanded rapidly in the past several years, these numbers demonstrate that the amortized cost per successful attack is low when a large enough net is cast.

7.1 Covert communication

Up to now, the network flow side channel has been used to make a binary determination of co-residency. Once co-residency has been determined, however, any manner of communication can take place over the channel. We are able to transmit a secret such as a small key or message with only a small amount of redundancy. We demonstrated this on our local ESXi testbed by creating a self-synchronizing CLIENT script that did not rely on out-of-band signaling from the FLOODER. The CLIENT's only prior knowledge is the size of the flood interval. The CLIENT reconstructs the signal by taking extremely rapid measurements and then searching for the local minima and maxima of the arrival patterns. These represent the 1's and 0's of the channel. It would also be possible to build more sophisticated communication protocols such as *Cloak* over this channel [32].

In the trial, the CLIENT initiated a TCP session with the SERVER and awaited a 2,048 bit message from the FLOODER. The first 10 s of the ensuing message is pictured in Fig. 10. Our CLIENT was able to decode the message with 100 % accuracy. As discussed by Cabuk et al. [12], the efficacy of an IP-based covert channel can be affected by contention noise in the channel and jitter in packet timings, which can lead to a loss of synchronization. Error-correcting codes, self-synchronizing codes, and phase-locked loops can be used to mitigate these issues. In our investigation, we included a 16-bit checksum for every 64-bit block transmitted by the FLOODER. This allowed the CLIENT to detect and recover from misreads in the watermark signal. This led to a total

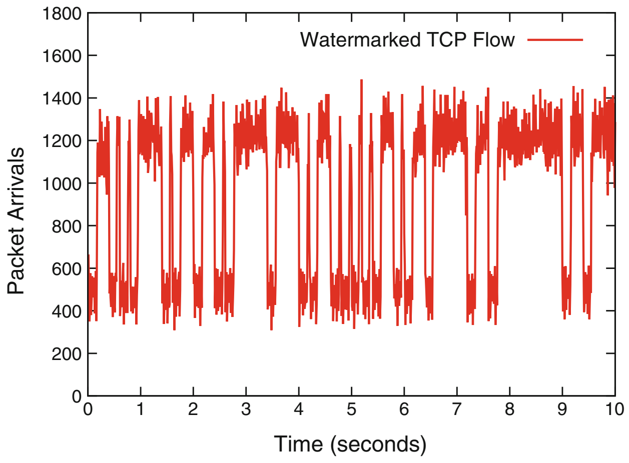


Fig. 10 A decodable side channel message created with co-resident watermarking. FLOODER activity in an interval reduces the CLIENT–SERVER connection bandwidth, signaling the transmission of a 1 bit

transmission of 2,560 bits. This required 10 min and 40 s of observed network flow, leading to a 4.00 bps side channel throughput. This bit rate compares favorably with other I/O-based covert channels [37]. If the participants possess outside knowledge about hardware and hypervisor configurations, they could further increase the bandwidth of the channel by decreasing the measurement size and reducing the wait time between sent bits. Additionally, more advanced error-correcting mechanisms such as the use of Hamming codes can increase the channel efficiency.

One particularly useful application of this method could be embedding a message into a network flow so as to bypass filtration mechanisms such as a national Web filter. In such a case, the message sender could co-locate to a known-allowed server, at which point they could embed a message into the server's network flows. There are two main benefits to this approach. First, the message is effectively multicast to all visitors to the server, meaning that even if the message were detected, the intended target would not be revealed. Second, an interested party, through entirely legitimate traffic, can retrieve the message while retaining plausible deniability. Additionally, this method works with no cooperation of the known-allowed host.

7.2 Load measurement

Previous work has demonstrated that virtualization side channels can be used to measure co-resident server load [43]. We build on this work with co-resident watermarking, discovering more accurate traffic information about our target's business. We accomplish this by simply monitoring the throughput of the undisturbed CLIENT–SERVER TCP session. The key insight that a co-resident instance provides is the ability to filter out additional causes of performance variance that would otherwise lead to false inferences—namely network congestion and changes in the load of co-resident instances. A co-resident flow serves as a second data point that allows for an accurate perspective of the target instance's load.

To perform load measurement, the FLOODER instance first uses co-resident watermarking to confirm that it is co-resident to the target SERVER. It then becomes a regular Web server, and the CLIENT initiates a single TCP session with both the SERVER and FLOODER. The CLIENT is able to observe the ratio between the throughputs of the two flows to generate a traffic profile of the victim. Network congestion can be detected and ignored by the fact that, since both flows will usually share a network path, flows will be impacted equally by network conditions. Thus, the ratio between the flows will remain constant. Changes in the load of other customers' virtual machines, or other noise introduced by the cloud datacenter, can be similarly disregarded. They affect both the SERVER and FLOODER flows equally, and therefore, the ratio between the flows will not change. The only scenario that changes the ratio between the CLIENT–SERVER and CLIENT–FLOODER connections is when the SERVER's load changes.

To demonstrate this behavior, we executed proof-of-concept trials on our local Xen testbed. The CLIENT initiated a single TCP session with the SERVER and FLOODER and then performed rapid measurements on both flows. Next, different load events were introduced and observed. For the first trial, pictured in Fig. 11, an increasing number of Web requests were issued from another host on the local network in 10-s intervals. The CLIENT calculated exponentially weighted moving averages of the two flows' packet arrivals and then took the ratio of the two. It can be observed that the SERVER-to-FLOODER throughput ratio decreases linearly, and basic system profiling techniques would allow the CLIENT to estimate the number of visitors to the victim instance. In the second trial, pictured in Fig. 12, Web requests are instead issued to other co-resident virtual machines. Every 22.5 s, 10 TCP sessions were initiated with a previously inactive virtual machine. In this scenario, the SERVER-to-FLOODER ratio remains roughly constant as both flows are adversely but proportionately affected. The increasing instability of the TCP flow may also serve as a second indicator of extreme load on the physical cloud node.

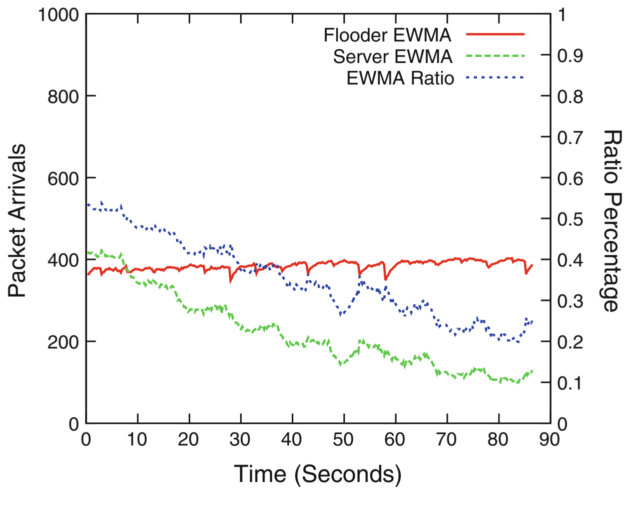


Fig. 11 Load Measurement Case 1: When SERVER load increases, decreasing the SERVER's per-connection performance, the ratio between the CLIENT–SERVER and CLIENT–FLOODER flows changes, allowing an attacker to profile the target SERVER

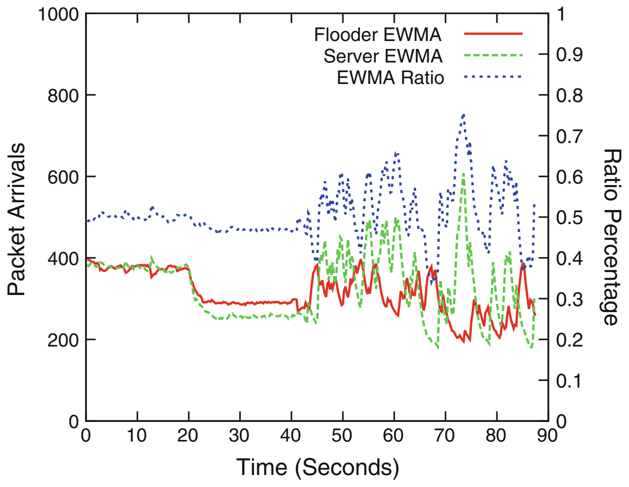


Fig. 12 Load Measurement Case 2: When additional guest VMs are introduced that decrease SERVERperformance, the ratio between the CLIENT–SERVER and CLIENT–FLOODER flows remains constant, allowing an attacker to filter out system noise

8 Toward detection avoidance

Up to this point, we have not considered the detectability of co-resident watermarking. *Invisibility* is a critical pursuit in all network flow watermarking schemes; if the attack is detected, the target can take a variety of countermeasures to frustrate the efforts of the watermarker. In the case of co-resident watermarking, we must also consider the possibility of anomaly detection systems put in place by the cloud administrator, which would be likely to monitor network activity. If the watermarking attempt is detected, the attacker's instances could be flagged as malicious or possibly terminated.

A serious limitation of the scheme proposed in Sect. 4, in which a co-resident VM launches large amounts of UDP probe traffic, is that it is easily detectable. Administrators, noticing multi-gbps UDP flows, could perform cursory packet inspection to establish that the probe traffic served no productive purpose. Worse yet, even the watermark target would be able to detect the attack. One way to accomplish this would be to monitor the timing of TCP packet acknowledgments, which would also contain the delay signals of the attack and could therefore be used to guess the watermark parameters and decode the signal. Below, we explore how an attacker might be able to overcome these limitations by impersonating the inherent noise of cloud datacenters.

8.1 Nature of cross-VM delay

Under regular circumstances, whether or not it is possible to create an invisible network flow watermark is an open question [11,20,34]. This is due largely to the fact that any anomalous delay, even small amounts, is observable by the watermark target. However, our domain-specific watermark scheme possesses several advantages over previous network flow watermarking attack models. First, in the absence of costly SLAs, cloud users must necessarily accept unpredictable delays, sometimes to the point of serious degradation of VM performance [5]. Due to limited ability to make inferences about co-resident VMs, identifying anomalies from the customer vantage point is also difficult. Additionally, hypervisors feature work-conserving configurations that allow customers to make use of free resources. Public clouds often enable these configurations in order to maximize efficiency at the cost of isolation, making resource contention between customers an inherent part of cloud usage.

We propose that this acceptance of delay, along with other out-of-band knowledge, can be used to create arbitrarily subtle watermarking schemes within the compute cloud domain. The heterogeneous nature of cloud use makes inference about malicious network activity extremely difficult. Consider the behavior of a legitimate cloud-based Web service. Such a service will provision exactly the number of VMs required to handle its current business level. In a steady state, we would therefore expect each instance to be highly utilized. If the business level drops, the service will terminate instances until all machines are again well saturated. If business suddenly spikes, the service will automatically provision more instances or migrate existing instances onto larger VMs. A variety of automated management solutions exhibiting this behavior have already been proposed by industry and the academic literature [46,53]. This leaves us with intuition that (i) instances are generally highly utilized, and (ii) spikes in traffic will occasionally overload user instances until more are provisioned and the load is rebalanced.

8.2 Design for detection avoidance

These observations are sufficient to create a co-resident watermark signature using seemingly innocuous customer behavior. We created a new scheme that leverages this insight, replacing conspicuous UDP probe traffic with innocuous TCP sessions. This new attack is pictured in Fig. 13. Using

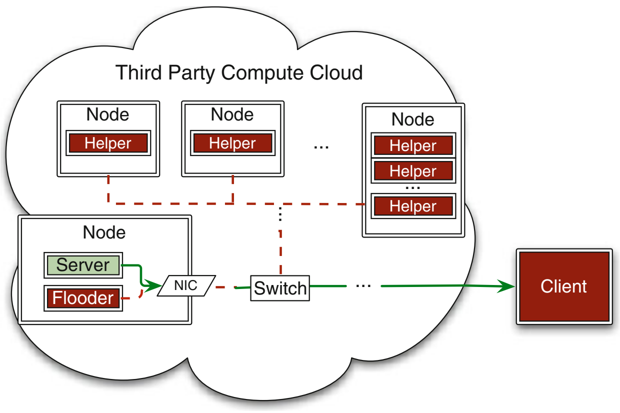


Fig. 13 Attack Model for TCP-based, detection-avoiding co-resident watermarking

out-of-band signaling, the CLIENT instructs a fleet of *Flood Helper* VMs, referred to henceforth as HELPERS, to initiate HTTP sessions with the FLOODER. To a system administrator, this activity appears to be purposeful communication between two different cloud-based services. The CLIENT instructs HELPERS to keep the FLOODER at a moderate workload. When it is time to send a signal, the CLIENT calls on more HELPERS to saturate the FLOODER, leading to a subsequent drop in SERVER throughput. These two FLOODER states, moderately utilized and saturated, represent the d_+ and d_- signals of our original scheme. The CLIENT then proceeds to monitor packet arrivals by interval and perform statistical analysis as before. From an administrator perspective, this appears as a sudden spike in business for an innocent customer. When the FLOODER returns to a reasonable workload, it appears that load has been rebalanced across more virtual machines; and in fact, the attacker will be provisioning more instances in order to conduct more probes. At no point is the activity of the FLOODER conspicuous.

Even while imitating a Web service, the attacker's activities could be detected by discovering a timing pattern within the observed delay. In order to frustrate attempts at discovering the existence of a watermark attack, we randomized the parameters of the watermark: signal length, signal strength, and wait between signals. The CLIENT sleeps for a random length of time before signaling, then selects a random signal length, and finally requests the signal from a random subset of all available HELPERS. As a result, the target and administrator are no longer able to guess the parameters of the watermark scheme, significantly increasing the complexity of proving the existence of the attack.

This new TCP-based scheme can be adjusted to the needs of the attacker, who faces a trade-off between efficiency and detectability. For cloud environments with minimal administrative supervision, the scheme can be parameterized to be as efficient as the original scheme explored earlier in this work. Alternately, the scheme can be further adjusted

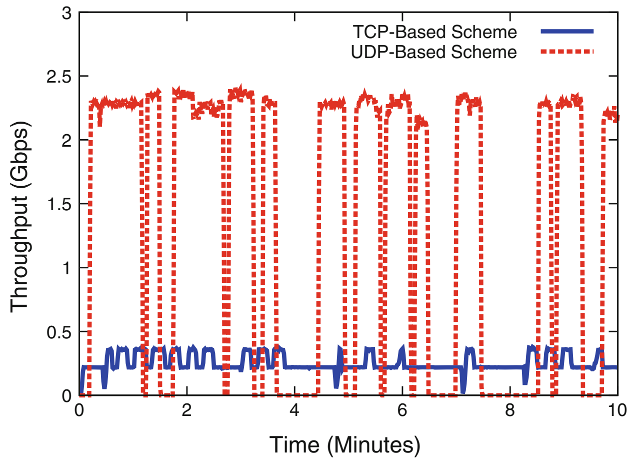


Fig. 14 Activity footprint of FLOODER in the two proposed co-resident watermarkingschemes

to better emulate the behavior of a particular cloud behavior, such as the auto-scaling of Amazon EC2's CloudWatch service.

8.3 Results with detection avoidance

The results for the two watermark schemes are contrasted in Fig. 14, which shows the outbound network activity of the co-resident FLOODER. It can be observed that the throughput of the FLOODER in this new scheme is significantly reduced. The flooding network activity is now HTTP traffic, which is unlikely to arouse the suspicions of cloud administrators. Moreover, the delta in flooding activity between the d_+ and d_- signal is now approximately 0.2 Gbps, instead of 2.3 Gbps, and the FLOODER is always engaging in some activity. Even with this new scheme, co-resident watermarking is still extremely effective. We are able to confirm co-residency in just 10 measurements on average. Our implementation verified co-location in less than 2 min, with an average wait between signals of 10 s and an average signal length of 10 s.

The ability to trigger resource contention with TCP, a protocol known for its fairness, requires further explanation. Fig. 14 shows us that that the total amount of flooding activity has been greatly diminished. This is because the TCP flows between the FLOODER and HELPERS are good network citizens, sharing more of the outbound bandwidth with the SERVER's flows. However, we also know that TCP will greedily increase its sending rate until it reaches an optimal bandwidth for the given network path. We showed in Sect. 6.6 that the bottleneck in our network path is likely to be the outbound interface of the cloud node. As a result, increased contention at the network interface will trigger the TCP congestion control mechanism. When more FLOODER connections are introduced, each of the existing TCP sessions back off, creating a measurable delay in the CLIENT–SERVER connection. TCP-based co-resident watermarking can therefore be used in any environment in which the UDP-based scheme was previously successful.

9 Limitations

The co-resident network flow represents a versatile side channel within the cloud. There are several defenses against the attack that we have explored in this work; one of them is restrictive network bandwidth caps, which was the mechanism that prevented us from successfully launching co-resident watermarkingon Amazon EC2 [48]. We explore the effectiveness and the economic drawbacks of various countermeasures below:

1. The most obvious defense is to provide each virtual machine instance with a dedicated path out of its physical host. Our approach is dependent on network flow multiplexing at the hypervisor and network interface card. While this defense would be 100 % effective, provisioning dedicated hardware is orthogonal to the purpose of cloud computing, which depends on the sharing of devices to provide low cost compute resources.
2. We also observed that co-resident watermarkingcan be thwarted by imposing bandwidth caps, conducting traffic shaping, or under provisioning of instances relative to the network transmission speed of their physical host. In fact, we confirmed the presence of a 300 Mbps transmission cap on EC2 small instances, making it significantly more difficult to launch our attack on their service [48]. Unfortunately, these techniques open up cloud providers to the possibility of wasted resources. Moreover, they are not foolproof defenses. In the case that VM density is high enough to avoid waste, network contention is reintroduced and co-resident watermarkingonce again becomes practical. Studies point to a rapid increase in VM density [19], indicating that network communication bottlenecks could become increasingly common in EC2.
3. It may also be possible that new, virtualization-aware hardware can address and close this side channel. However, our experience with the Intel 82599 ES controller indicates that, for manufacturers, addressing virtualization's performance challenges is paramount. In fact, the isolation mechanism in the *ixgbe* driver allowed our FLOODER to throttle the SERVER's throughput at will. SR-IOV and other pass-through technologies increase the exposure of underlying hardware and may continue to increase the effectiveness of side channels unless isolation is made a primary design goal.
4. Another possible defense would be to apply the random scheduling strategy previously employed in cache

measurements [25] to do outbound packet scheduling. While this would be effective, it could trigger TCP congestion control [47] and degrade performance across all virtual machines. Our attack is different from cache-based attacks in that the protocol and expected behavior act as an enforcement mechanism to prevent excess non-determinism from marring our data. Additionally, the proposed defenses would break certain network-related aspects of resource scheduling by the hypervisor.

The problem we illustrate is inherent in resource sharing and is particularly essential to the cloud computing economy, which is based on maximizing resource utilization. By launching co-resident watermarkingon three of the major virtual hypervisor platforms, we have demonstrated that systemic resource-sharing vulnerabilities are not unique to a particular virtualization initiative. Moreover, we have demonstrated that delegating resource management to the hardware level can further exacerbate this problem. A consequence of this work is the need for hardware drivers that extend the isolation guarantees of the hypervisor, sacrificing minimal performance in exchange for increased privacy.

10 Related work

10.1 Cloud side channel attacks

A variety of cross-VM side channels have been demonstrated in the academic literature. Deficiencies in performance isolation, similar to those leveraged in this work, have been exploited for a variety of purposes. Noting that cache and network utilization are often contested between VMs, a resource freeing attack (RFA) has been proposed that allows a greedy customer to manipulate the performance of co-resident VMs by shifting their resource bottlenecks [48]. This work operates under a similar attack model as our own, targeting public network cloud services and manipulating VMs from a helper host. However, where RFA is a performance enhancement strategy for the cloud, co-resident watermarking is a method of information extraction.

Cache-based side channel attacks, in which timing differences in access latencies between the cache and main memory are exploited, have attracted the most attention for cloud computing. Most notably, Zhang et al. [58] demonstrated that the machine instructions of a co-resident VM can be recovered from shared L1 caches, permitting the reconstruction of secret keys in the circumstance that they influence the code path of a decryption routine. Ristenpart et al. [43] showed that cache usage can be examined as a means to measure the activity of other instances co-resident with the attacker. Furthermore, they demonstrated that they can detect co-residency with a victim's instance if they have information about the instance's computational load. In contrast, Zhang et al. [57] utilized cache-based side channels as a defensive mechanism. Their scheme works by measuring cache footprints for evidence of other VMs. Leveraging this scheme, they can challenge correct functionality on the part of the cloud provider and discover other unanticipated instances sharing the same host.

Bowers et al. [8] have proposed use of a different network timing side channel in order to challenge fault tolerance guarantees in storage clouds. This work measures the response time of random data reads in order to confirm that a given file's storage redundancy meets expectations. This scheme can be used to detect drive-failure vulnerabilities and expose cloud provider negligence. We intend to investigate the applicability of storage cloud co-resident watermarking in future work.

10.2 Hypervisor security

Raj et al. [41] proposed two other mechanisms for preventing cache-based side channels, cache hierarchy aware core assignment, and page-coloring-based cache partitioning. The former groups CPU cores based on last level cache (LLC) organization and checks whether such organization has any conflict with the SLA of the clients. The latter is a software method that monitors how the physical memory used by applications maps to cache hardware, grouping applications accordingly to isolate clients. Another effective defense against cache-based side channels is changing how caches assign memory to applications, such as non-deterministic caches [26]. Non-deterministic caches control the lifetime (decay interval) of cache items. By assigning a random decay interval to cache items, the cache behavior becomes non-deterministic, and hence, side channels cannot exploit it. Work in performance isolation in Xen can also lead to added security benefits [21].

Other work aims to combat virtualization vulnerabilities by reducing the role and size of the hypervisor. Most drastically, Keller et al. [25] eliminate a large attack surface by proposing the near elimination of the hypervisor. This is achieved through preallocation of resources, limited virtualized I/O devices, and modified guest operating systems. While this approach inarguably reduces the likelihood of exploitable implementation flaws in the virtualization code base, it necessarily places VMs closer to underlying hardware. Intuitively, this can only increase the bandwidth of the isolation-compromising side channel that we explore in this work. Other proposals reduce the hypervisor attack surface by considering only specific virtualization applications such as rootkit detection or integrity assurance for critical portions of security-sensitive code [35,45] or by distributing administrative responsibilities across multiple VMs

[10]. We do not consider these systems in our work because they are not intended for the third-party compute cloud model.

10.3 In-the-wild exploits

The Xen and VMWare communities have discovered only a handful of privilege escalation exploits. The presence of such attacks greatly incentivizes efficient co-resident detection schemes. An early version of Xen 3 included a bug that caused domU grub files to be executed without protection in dom0 [15]. The exploit allowed users to craft malicious grub.conf files that led to arbitrary code execution in the administrative domain. Earlier versions of Xen included a buffer overflow error that allowed specially crafted disk images to execute code in dom0 [16]. In 2008, a bug was discovered in the folder-sharing feature of some VMWare product lines that allowed for unprivileged user code to be executed by the vmx process [49]. More recently, a paging function in Linux kernels 2.6.35.2 and earlier allowed for a guest domain to perform a memory exhaustion attack on the system [17]. Lastly, in 2012, partial source code for VMWare's ESX hypervisor leaked [9], and while no exploits have been directly attributed to this leak yet, such incidents increase the risk of compromise.

11 Conclusion

In this work, we have leveraged active traffic analysis techniques as a means of determining co-residency of instances in cloud environments. We show that our co-resident watermarkingscheme can be used to make a determination of co-residency in under 10 s for a given probe in the cloud. We demonstrate the feasibility of this attack by deploying it in multiple production cloud environments in geographically disparate locations and running a diverse set of hypervisors. We are able to interpose a covert channel on our target's network flow and show means of performing passive attacks such as load measurement against the cloud-based target. We go on to create a detection-avoidance strategy that masks co-resident watermark signaling within innocuous cloud customer activities. These investigations further demonstrate the ramifications of multiplexing hardware in virtualized environments, demonstrating a need for cloud hardware that is performed without introducing undesired side effects.

Acknowledgments We would like to thank Allen D. Malony, Chris Hoge, and the ACISS staff for their assistance and support. Through our use of Futuregrid, this material is based upon work supported in part by the National Science Foundation Under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid Test-bed." and Grant CNS-1118046.

Appendix A Hypervisor scheduling

A.1 Xen

Xen is a popular type I virtual hypervisor that allows multiple operating systems to share hardware through the use of abstracted paravirtualized interfaces. Xen separates policy and mechanism by having its hypervisor's device scheduler providing only the most basic operations. Higher-level scheduling algorithms are the responsibility of the `domain 0` (dom0) guest operating system, which acts as an administrator and has access to a hypervisor control interface. In this way, Xen's schedulers implement fair scheduling of resources for guest domains (domU).

Xen schedules domain CPU utilization using the Borrowed Virtual Time (BVT) algorithm [4]. BVT has a special low-latency wake-up mechanism that temporarily favors domains that have just received an event. This allows for the effect of virtualization to be minimized for services such as TCP that require accurate round-trip time measurements. Xen provides real time, virtual time, and wall-clock time to guest domains to ensure correct sharing of time slices for their own applications.

For networking, Xen provides virtual network interfaces (VIFs) that attach to a virtual firewall-router (VFR). Each VIF in dom0corresponds to an interface that is visible in a domU. The VFW performs services such as demultiplexing received packets based on destination IP and port. VIFs emulate physical network interface cards by providing transmit and receive I/O rings. Guest domains transmit packets by enqueueing packets onto the transmit ring and receive packets by exchanging unused page frames for each packet dequeued from the receive ring. Each domUpacket passes through dom0on its way to or from the physical interface. Xen packet scheduling is a simple round robin.

Recent work has shown that the Xen hypervisor introduces considerable packet transmission delays under heavy network usage, adding on the order of 100 ms to round-trip times [52], limiting network throughput to as little as 2.9 Gbps [42]. A great deal of this delay is introduced through the packet needing to pass through dom0. The use of paravirtualized interfaces and software network bridges also adds delay when compared to hardware virtualization. As our work seeks to inject as much delay into a network flow as possible, we made use of these artifacts of the Xen hypervisor in addition to the limitations of underlying physical devices. However, we demonstrate in Section 6 that our scheme is also effective on lightweight hypervisors.

A.2 VMWare ESXi

VMWare ESXi is another operating system-independent hypervisor that allows multiple virtual machines to share physical hardware. Unlike Xen, ESXi eliminates the privileged guest partition and runs all management and infrastructure services directly from a micro-kernel (VMkernel). The reduced footprint of the ESXi hypervisor creates a smaller surface for vulnerability. ESXi implements a proportional-share- based algorithm for domain CPU utilization scheduling. With this mechanism, scheduling decisions are prioritized based on the ratio of the consumed CPU resources to the entitled resource limit of each virtual CPU (vCPU). Lower ratios are given higher priority, thus giving vCPUs with greater resource needs higher precedence. To increase performance, ESXi also implements relaxed coscheduling with symmetric multi-processing, which allows multiple threads or processes to be executed in parallel over multiple physical CPUs. Packet scheduling relies on a simple round-robin method.

A.3 KVM

KVM is a type 2 hypervisor for Linux platforms and is designed to reuse as much of the underlying Linux infrastructure as possible. With KVM, each VM is treated as a process and is scheduled using the default Linux scheduler, which is the Completely Fair Scheduler (CFS) [22]. CFS tracks the *virtual runtime* of each process, which is the time allocated to each task to access the CPU. Smaller virtual runtimes result in higher priority. CFS also implements *sleeper fairness*, in which waiting processes are treated as if they were on the run queue, so they receive a comparable share of CPU time when they need it.

In contrast to many other schedulers, CFS uses a time-ordered red-black tree instead of a queue to maintain waiting processes. Processes with higher priority (lower virtual runtime) are placed on the left side of the tree, and processes with lower priority (higher virtual runtime) are stored in the right side. The scheduler selects the leftmost node to run, and then to maintain fairness, the process's execution time is added to the virtual runtime and the process is reinserted into the tree. This tree is self-balancing, and tree operations run in O(*log n*) time.

Appendix B Virtualization-aware devices

As the number of VMs operating on a system increases, network performance can drastically decrease in hypervisors that mediate network access with an administrative domain. The traditional single CPU core handling received packets is not sufficient to service the number of incoming packets on a 10 GB Ethernet connection. Virtualization-aware hardware can be employed to mitigate these bottleneck risks and increase networking efficiency. Two such hardware specifications are Virtual Machine Device Queues (VMDq) [13] and Single Root I/O Virtualization (SR-IOV) [18].

VMDq is a silicon-level technology that alleviates network traffic bottlenecks by offloading packet-sorting responsibility from the hypervisor to the NIC. Within the NIC, there exist unique queues for each VM to receive their assigned packets. Relieving the VMM of network traffic sorting allows more CPU cycles to be granted to the VMs themselves. Both Xen and ESXi support VMDq technology with baked-in software provided for additional efficiency. Xen implements a new protocol for I/O channels, called Netchannel2, which reduces I/O bottlenecks in dom0 by performing packet sorting within the receiving domain instead of in dom0. ESXi's VMDq support comes from NetQueue, a similar software package.

SR-IOV is a specification that allows physical I/O devices to present themselves to the host as multiple virtualized I/O devices, allowing for direct access to PCI interfaces. This is especially impactful when considering network access in Xen, as it eliminates the need for dom0 to be involved in copying packet buffers from the guest OS. Since each domU has access to its own PCI virtual function, SR-IOV also provides individual queues for each VM. Arriving packets are sorted into these queues by the physical device based on their destination, then are copied directly to the guest OS memory using direct memory access (DMA). VMWare's implementation of SR-IOV, called VMDirectPath, permits direct-assignment technologies to achieve device sharing.

References

1. Amazon EC2 Service Level Agreement. http://aws.amazon.com/ec2-sla/
2. Amazon. Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2/
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., et al.: Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, University of California, Berkeley (2009)
4. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield A.: Xen and the art of virtualization. In: Proceedings of 19th ACM Symposium on Operating Systems Principles, SOSP '03, New York, pp. 164–177. ACM (2003)
5. Barker, S., Shenoy P.: Empirical evaluation of latency-sensitive application performance in the cloud In: Proceedings of 1st SCM SIGMM Conference on Multimedia Systems, MMSys '10, New York, pp. 34–46. ACM (2010)
6. Bernstein D.J.: Cache-timing attacks on AES. Compute (2005)
7. Blum, A., Song, D., Venkataraman, S.: Detection of interactive stepping stones: algorithms and confidence bounds. In: Proceedings of Recent Advances in Intrusion Detection (RAID) (2004)
8. Bowers, K.D., van Dijk, M., Juels, A., Oprea, A., Rivest R.L.: How to tell if your cloud files are vulnerable to drive crashes. In: CCS '11: Proceedings of 18th ACM Conference on Computer and Communications Security, Chicago, pp. 501–514 (2011)

9. Brodkin J.: VMware confirms source code leak, LulzSec -affiliated hacker claims credit. http://arstechnica.com/business/news/2012/04/vmware-confirms-source-code-leak-lulzsec-affiliated-hacker-claims-credit.ars
10. Butt, S., Lagar-Cavilla, H.A., Srivastava, A., Ganapathy V.: Self-service cloud computing. In: Proceedings of 2012 ACM Conference on Computer and Communications Security, Raleigh (2012)
11. Cabuk, S., Brodley, C.E., Shields C.: Ip covert timing channels: design and detection. In: Proceedings of 11th ACM Conference on Computer and Communications Security, CCS '04, New York, pp. 178–187. ACM (2004)
12. Cabuk, S., Brodley, C.E., Shields C.: IP Covert Channel Detection. ACM Trans. Inf. Syst. Secur. (TISSEC) **12**(4): 1–29 (2009)
13. Chinni, S., Hiremane, R.: Virtual machine device queues. White paper, Intel Corporation (2007)
14. Coskun, B., Memon, N.: Online sketching of network flows for real-time stepping-stone detection. In: Proceedings of 2009 Annual Computer Security Applications Conference, ACSAC '09, Washington, pp. 473–483. IEEE Computer Society (2009)
15. CVE-2007-4993. pygrub (tools/pygrub/src/grubconf.py) in xen 3.0.3. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4993
16. CVE-2007-5497. Multiple integer overflows in libext2fs. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5497
17. CVE-2010-2240. The do_anonymous_page function in mm/memory.c. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2240
18. Dong, Y., Yu, Z., Rose, G.: SR-IOV networking in Xen: architecture, design and implementation. In: Proceedings of First Conference on I/O Virtualization, WIOV'08, Berkeley, p. 10. USENIX Association (2008)
19. Gamage, S., Kangarlou, A., Kompella, R.R., Xu, D.: Opportunistic flooding to improve TCP transmit performance in virtualized clouds. In: Proceedings of 2nd ACM Symposium on Cloud Computing, SOCC '11, New York, pp. 1–14. ACM (2011)
20. Gianvecchio, S., Wang, H.: Detecting covert timing channels: an entropy-based approach. In: Proceedings of 14th ACM Conference on Computer and Communications Security (CCS'07), Alexandria (2007)
21. Gupta, D., Cherkasova, L., Gardner, R., Vahdat, A.: Enforcing performance isolation across virtual machines in Xen. In: Middleware (2006)
22. Habib, I.: Virtualization with KVM. Linux J. 166: 8(2008)
23. Houmansadr, A., Borisov, N.: SWIRL: a scalable watermark to detect correlated network flows. In: Proceedings of 18th ISOC Symposium on Network and Distributed Systems Security (NDSS '11), San Diego (2011)
24. Houmansadr, A., Kiyavash, N., Borisov, N.: RAINBOW: a robust and invisible non-blind watermark for network flows. In: Proceedings of 16th Network and Distributed System Security Symposium (NDSS'09) (2009)
25. Keller, E., Szefer, J., Rexford, J., Lee, R.B.: Eliminating the hypervisor attack surface for a more secure cloud. In: Proceedings of ACM Conference on Computer and Communications, Security (CCS'11) (2011)
26. Keramidas, G., Antonopoulos, A., Serpanos, D., Kaxiras, S.: Non deterministic caches: a simple and effective defense against side channel attacks. Design Autom. Embed. Syst. **12**: 221–230 (2008)
27. Kiyavash, N., Houmansadr, A., Borisov, N.: Multi-flow attacks against network flow watermarking schemes. In: Proceedings of 17th USENIX Security Symposium, San Jose (2008)
28. Kocher, P.C.: Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems. In: CRYPTO, pp. 104–113 (1996)
29. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: CRYPTO, pp. 388–397 (1999)
30. Kutch, P.: PCI-SIG SR-IOV Primer. Technical report, Intel Corporation (2011)
31. Law, A.M., Kelton, D.W.: Simulation Modeling and Analysis. McGraw-Hill, Boston (2000)
32. Luo, X., Chan, E., Chang, R.: Cloak: A ten-fold way for reliable covert communications. In: Proceedings of European Symposium on Research in Computer Security ESORICS (2007)
33. Luo, X., Zhang, J., Perdisci, R., Lee, W.: On the secrecy of spread-spectrum flow watermarks. In: Proceedings of European Symposium on Research in Computer Security ESORICS (2010)
34. Luo, X., Zhou, P., Zhang, J., Perdisci, R., Lee, W., Chang, R.K.C.: Exposing invisible timing-based traffic watermarks with BACKLIT. In: Proceedings of 27th Annual Computer Security Applications Conference, ACSAC '11, Orlando (2011)
35. McCune, J.M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., Perrig, A.: TrustVisor: efficient TCB reduction and attestation. In: Proceedings of 2010 IEEE Symposium on Security and Privacy, Oakland (2010)
36. Murdoch, S., Danezis, G.: Low-cost traffic analysis of Tor. In: Proceedings of 2005 IEEE Symposium on Security and Privacy. Oakland (2005)
37. Okamura, K., Oyama, Y.: Load-based covert channels between Xen virtual machines. In: Proceedings of 2010 ACM Symposium on Applied Computing, SAC '10, Sierre (2010)
38. Peng, P., Ning, P., Reeves, D.S.: On the secrecy of timing-based active watermarking trace-back techniques. In: Proceedings of 2006 IEEE Symposium on Security and Privacy, Oakland (2006)
39. Percival, C.: Cache missing for fun and profit. In: BSDCan (2005)
40. Pettitt, A.N., Stephens, M.A.: The Kolmogorov–Smirnov goodness-of-fit statistic with discrete and grouped data. Technometrics **19**(2), 205–210 (1977)
41. Raj, H., Nathuji, R., Singh, A., England, P.: Resource management for isolation enhanced cloud services. In: Proceedings of 2009 ACM Workshop on Cloud Computing Security, CCSW '09, Chicago (2009)
42. Ram, K.K., Santos, J.R., Turner, Y., Cox, A.L., Cox, A.L., Rixner, S.: Achieving 10 GB/s using Xen para-virtualized network drivers. Xen Summit (2009)
43. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get off of my cloud: exploring information leakage in third-party compute clouds. In: CCS'09: Proceedings of 16th ACM Conference on Computer and Communications Security, Chicago (2009)
44. Schad, J., Dittrich, J., Quiané-Ruiz, J.-A.: Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proc. VLDB Endow. **3**(1–2), 460–471 (2010)
45. Seshadri, A., Luk, M., Qu, N., Perrig, A.: SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In: SOSP'07: Proceedings of 21st ACM Symposium on Operating Systems Principles, Stevenson (2007)
46. Singh, A., Korupolu, Aameek M., Mohapatra, D.: Server-storage virtualization: integration and load balancing in data centers. In: Proceedings of 2008 ACM/IEEE Conference on Supercomputing, Austin (2008)
47. Stevens, W.R.: TCP/IP Illustrated: The Protocols, vol. 1. Addison-Wesley Longman Publishing Co. Inc., Boston (1993)
48. Varadarajan, V., Kooburat, T., Farley, B., Ristenpart, T., Swift, M.M.: Resource-freeing attacks: improve your cloud performance (at Your Neighbor's Expense). In: Proceedings of 2012 ACM Conference on Computer and Communications Security, Raleigh (2012)
49. VMSA-2008-0008. Updates to VMware workstation, VMware player, VMware ACE, VMware fusion resolve critical security issues. http://www.vmware.com/security/advisories/VMSA-2008-0008.html
50. Wang, X., Chen, S., Jajodia, S.: Network flow watermarking attack on low-latency anonymous communication systems. In: Proceed-

ings of 2007 IEEE Symposium on Security and Privacy, Oakland (2007)

51. Wang, X., Reeves, D.S.: Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In: Proceedings of 10th ACM Conference on Computer and Communications Security, CCS '03, New York, pp. 20–29. ACM (2003)
52. Whiteaker, J., Schneider, F., Teixeira, R.: Explaining packet delays under virtualization. SIGCOMM Comput. Commun. Rev. **41**: 38–44 (2011)
53. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Black-box and gray-box strategies for virtual machine migration In: Proceedings of 4th USENIX Conference on Networked Systems Design and Implementation, Cambridge (2007)
54. Xu, Y., Bailey, M., Jahanian, F., Joshi, K., Hiltunen, M., Schlichting R.: An exploration of L2 cache covert channels in virtualized environments. In: Proceedings of 3rd ACM Workshop on Cloud Computing, Security (CCSW'11) (2011)
55. Yao, Y.: Network speed test (IPerf) in KVM (Virtio-net, emulated, vt-d). http://vmstudy.blogspot.com/2010/04/network-speed-test-iperf-in-kvm-virtio.html (2004)
56. Yu, W., Fu, X., Graham, S., Xuan, D., Zhao, W.: DSSS-based flow marking technique for invisible traceback. In: Proceedings of 2007 IEEE Symposium on Security and Privacy (2007)
57. Zhang, Y., Juels, A., Oprea, A., Reiter, M.K.: HomeAlone: Co-Residency Detection in the Cloud via Side-Channel Analysis. In: Proceedings of 2011 IEEE Symposium on Security and Privacy, Berkeley (2011)
58. Zhang, Y., Juels, A., Reiter, M.K., Reiter, M., Ristenpart, T.: Cross-VM side channels and their use to extract private keys. In: Proceedings of 2012 ACM Conference on Computer and Communications Security, Raleigh (2012)