Detecting Co-Residency with Active Traffic Analysis Techniques

Adam Bates, Benjamin Mood, Joe Pletcher,
Hannah Pruse, Masoud Valafar, and Kevin Butler
Oregon Systems Infrastructure Research and Information Security (OSIRIS) Lab
University of Oregon, Eugene, OR
{amb,bmood,pletcher,hpruse,masoud,butler}@cs.uoregon.edu

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 without costly underutilization of the physical machine. We evaluate co-resident watermarking under a large variety of conditions, system loads and hardware configurations, from a local lab 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 less than 10 seconds. We also show that passive load measurement of the target and subsequent behavior profiling is possible with this attack. Our investigation demonstrates the need for the careful design of hardware to be used in the cloud.

Categories and Subject Descriptors

K.6.5. [**Security and Protection**]: Unauthorized access

General Terms

Experimentation, Measurement, Security

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 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 amongst 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 methods of bypassing co-resident isolation in virtualization middleware, particularly through the L2 cache [37, 45, 47]. 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 [35].

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 inter-packet 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 amongst hypervisors used in clouds, our attack is implementation-independent. We can determine whether instances are co-resident in under 10 seconds and as few as 2.5 seconds 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 seconds.

- **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 previous work [37]. We also demonstrate the creation of a covert channel capable of transmitting 4 bps of information.

The rest of this paper is organized as follows. We provide a brief introduction to the issue of cloud co-residency in Section 2, and present the relevant concepts of active traffic analysis, particularly network flow watermarking, in Section 3. Section 4 presents a threat model and our co-resident watermarking encoding and decoding steps. In Section 5 we elaborate on the application of our scheme. Our attack is thoroughly evaluated in Section 6 under various conditions. Practical use scenarios are considered in Section 7 and countermeasures discussed in Section 8. Related work is considered in Section 9 before we conclude in Section 10.

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 subtler 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 though virtualization side channels is a danger that was first exposed by Ristenpart et al. [37]. 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

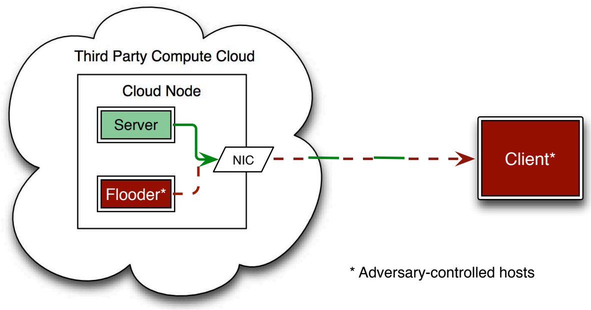


Figure 1: The attack model considered for co-resident watermarking. Two colluding hosts, the CLIENT and FLOODER, attempt communicate through the legitimate network flow of the SERVER.

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 [35], 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 [37], 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 [8, 9], 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 [20, 21, 42, 46]. 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 [5, 11, 28, 43], and compromising network anonymity services (e.g. TOR network) [24, 29].

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 [24]. However, recent work has shown that even the most advanced schemes do not possess the invisibility property [8, 17, 29]. As such, we do not pursue invisibility as a goal in this preliminary work, focusing instead on determining the efficacy and throughput of the co-resident network flow channel. We consider the unique advantages and challenges of developing an invisible version of our scheme in Section 8.1.

Our methodology also bears similarities to previous work on traffic analysis of Tor. Murdoch et al demonstrated that a corrupt Tor node can collude with a network server to extract information about the path of a Tor connection [31]. 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 [31] 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 to fuzz the results of these services and prevent co-residency detection. To their credit, the administrators in this cloud have also proactively applied patches that have all but eliminated popular hypervisor side channels such as the L2 cache. 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. We also assume that our victim is trusting of 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 [37]. 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 pre-determined 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 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 Figure 1.

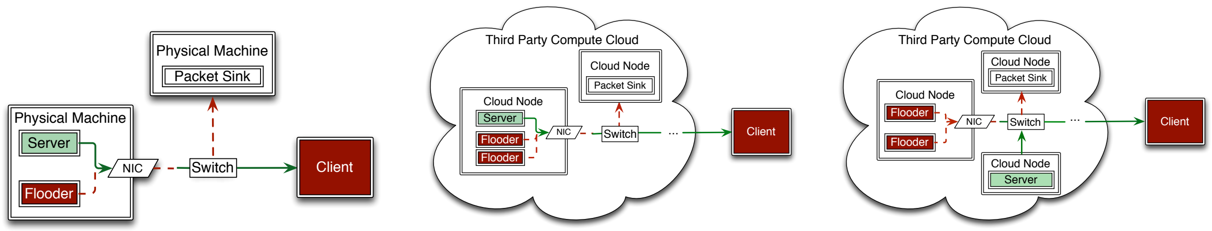
4.3 Signal Encoding

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 $(+d, -d)$ 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 hypervisor scheduling, network congestion, and virtualization-imposed artifacts. While these factors will not remain constant for any meaningful length of time [38], their effects can be filtered out by randomly selecting each w_i in sequence. In Section 6, we demonstrate that watermark signals can be decoded in spite of the presence of any of these factors.



(a) Local testbed. (b) Cloud, successful co-location. (c) Cloud, failed co-location.

Figure 2: Testbed topologies used in evaluation. Adversary-controlled hosts are shaded in red.

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 pre-negotiated co-resident activity representing $+d$ and $-d$. 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 modeled packet arrivals by a Poisson distribution [26], and employed the non-parametric Kolmogorov-Smirnov (KS) test for independence [34]. This statistical measure has been employed previously in other analysis of covert timing channels [17, 33]. 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.

An alternate non-parametric test that is better known for use with discrete distributions is the Pearson 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 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

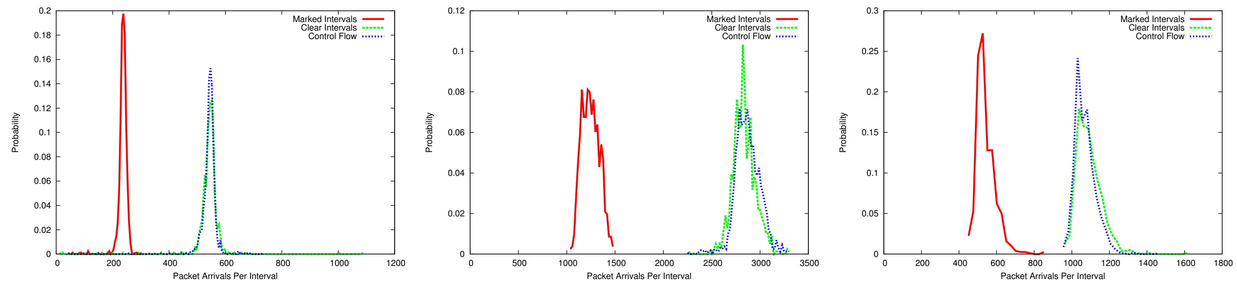
Our target instance, the SERVER, was a virtual machine running Apache 2. The CLIENT host initiated a TCP session with the SERVER, continuously re-requesting a 10MB file. To create more realistic web traffic conditions, we wrote a PHP script that simulated background noise on a server. The script conservatively 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 is 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 customer instances, 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 Section 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 250ms and then waited 2 seconds before signaling the flooder again. This was sufficient to ensure that our measurements were independent.

6. EVALUATION

We used a number of different testbeds to evaluate our approach, as shown in Figure 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



(a) Xen on local testbed. (b) VMWare ESXi on local testbed. (c) Xen over long network path.

Figure 3: Probability distributions for co-resident watermarking.

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 2GB 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.

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 8GB 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 Figure 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, 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 CLIENT initiated a single TCP session with the SERVER's `apache` process. The CLIENT then 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 Figure 3a. This figure and all others are based on 3200 total measurements that correspond to 13 minutes and 20 seconds 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 seconds 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 with 95% confidence. This is sufficient to declare that our instances are co-located.

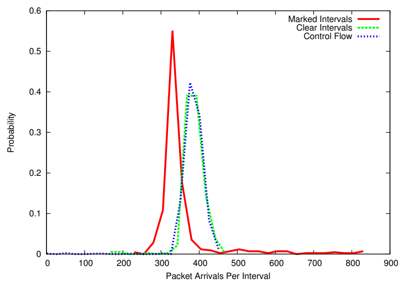
6.2 VMWare ESXi Hypervisor

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. ESXi lacks Xen's dom0 administrative domain and is therefore much more efficient at packet transmission. The results, shown in Figure 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 demonstrates the feasibility of co-resident watermarking on two of the major hypervisor families.

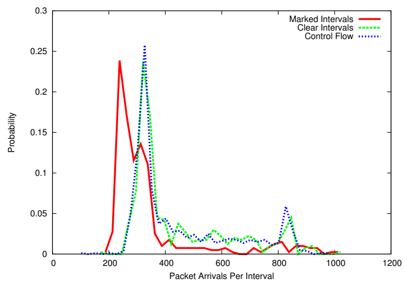
6.3 System Load

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.

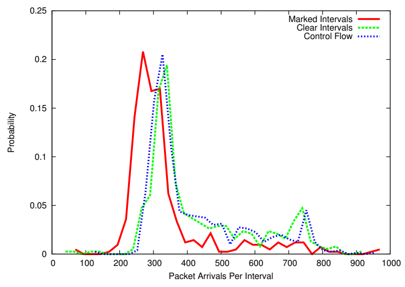
We repeated our standard trial with up to 3 GUESTs for a total of 5 instances on the machine. This load approached the maximum



(a) 1 additional GUEST.



(b) 2 additional GUESTs.



(c) 3 additional GUESTs.

Figure 4: Density functions for co-resident watermarking with increasing numbers of I/O bound web server guest instances.

Trial	Length	$KS_{+d,-d}$	p-val	Result
SERVER & FLOODER	2.5 *sec*	0.99	0.01	*Co-Res*
Add 1 GUEST	3.75 *sec*	0.78	0.05	*Co-Res*
Add 2 GUESTs	3.75 *sec*	0.91	0.01	*Co-Res*
Add 3 GUESTs	10 *sec*	0.49	0.05	*Co-Res*

Table 1: Results of tests in Xen as system load increases. Minimum flow lengths required to achieve 95% confidence are displayed.

capacity of our testbed. The results of these trials are pictured in Figures 4a-4c. As the number of GUESTs on the machine increase, we see distribution of the marked samples begin 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, show that we are able to quickly confirm co-residency with up to 5 guests on our local testbed.

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 Figure 3c. Once again, there is a no visible similarity between the clear and marked distributions. The watermark signature is still identifiable after just 2.5 seconds 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; we were only able to generate approximately 3.2 Gbps of traffic from a single FLOODER instance, falling well short of the 10 Gbps channels. This prevented us from injecting packet delay into the CLIENT-SERVER flow. Because we were only off by a small constant factor, we re-attempted the trial with multiple co-resident FLOODERs. This topology is pictured in Figure 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 [16], 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 [16, 36].

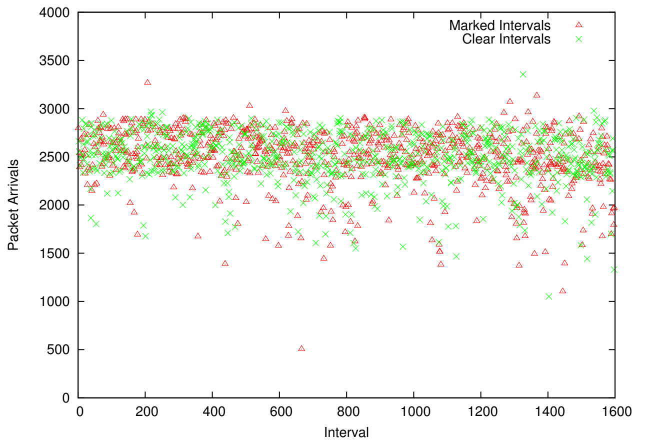
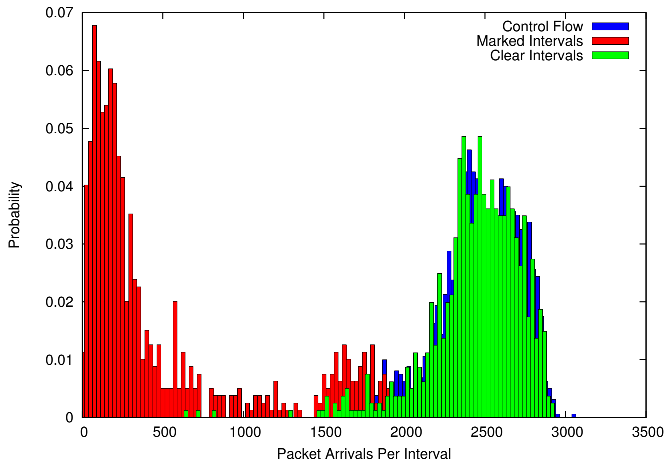
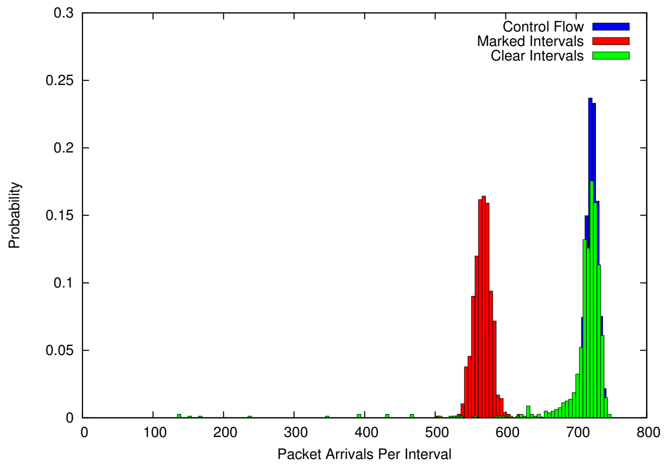


Figure 5: FLOODER activity does not significantly impact neighboring physical machines.

The results of these trials are visible in Figures 6a and 6b. 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 seconds 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 watermarking on a Futuregrid cloud. The KS test yielded a statistic of 0.97 with p-value of 0.02 after 2.5 seconds of observed flow. These tests demonstrate that our aurrent implementation is nearly practical for industrial compute clouds.



(a) University of Oregon's ACISS Cloud



(b) Futuregrid Sierra

Figure 6: Results from trials run on a industrially provisioned compute clouds.

6.6 Neighboring Instance False Positives

We have shown co-resident watermarking to 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 resultingly 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 watermarking is not susceptible to false positives, we configured a new topology on 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 Figure 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 Figure 5. 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 0.981 and p-value of 0.01 after 2.5 seconds of observed network flow. They are statistically similar enough to accept the null hypothesis that they were drawn from the same distribution.

6.7 Virtualization-Aware Hardware

As a preliminary investigation into the viability of hardware-level defenses against co-resident watermarking, we repeated our original Xen trial on an SR-IOV-enabled NIC. SR-IOV [25] 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 dom0 to be involved in copying packet buffers from the guest domain. 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 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.

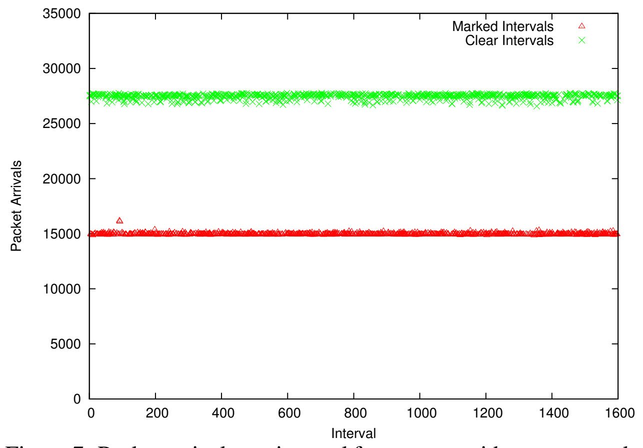


Figure 7: Packet arrivals per interval for our co-resident watermarking attempts against an SR-IOV-enabled network device. The unmarked traffic transmitted data at 0.83 Gbps, with the marked traffic at 0.41 Gbps.

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 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, reducing the bandwidth of the NIC while preserving the driver's behavior.

The results from this trial are shown in Figure 7. 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 ($\sim$ 0.17 Gbps). When the FLOODER is inactive, the SERVER is able to transmit at the highest possible rate ($\sim$ 0.33 Gbps). The KS test trivially rejects the null hypothesis. The FLOODER's ability to have such an impact indicates that, unlike some 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 carefully considered before their adoption.

7. ANALYSIS

We have demonstrated that co-resident watermarking is 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 executed a proof-of-concept execution 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 on 1684 targets with 1784 probes. At the current rate of $0.08 per hour for small Amazon EC2 instances, our attack would cost $1.01 and require 6 minutes 20 seconds per successful co-location. This estimate assumes 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 communications protocols such as *Cloak* over this channel [27].

In the trial, the CLIENT initiated a TCP session with the SERVER and awaited a 2048 bit message from the FLOODER. The first 10 seconds of the ensuing message are pictured in Figure 8a. Our CLIENT was able to decode the message with 100% accuracy. As discussed by Cabuk et al. [9], 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 allows the CLIENT to detect and recover from misreads in the watermark signal. This leads to a total transmission of 2560 bits. This required 10 minutes and 40 seconds of observed network flow, leading to a 4.00 bps side channel throughput. This bit rate is compares favorably with other I/O based covert channels [32]. 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.

7.2 Load Measurement

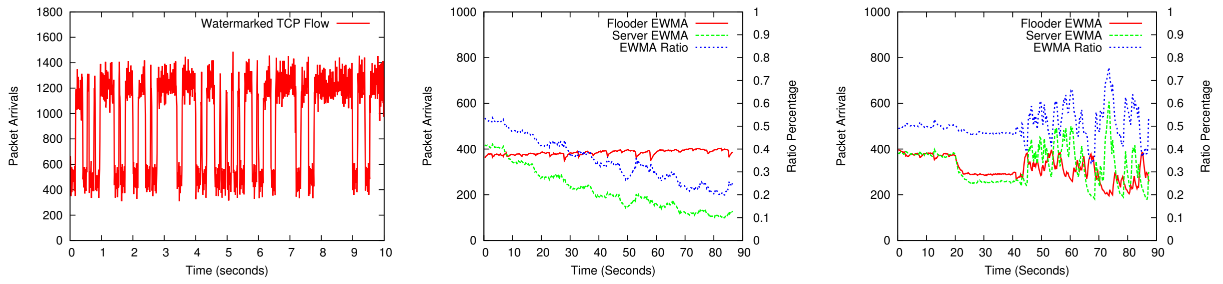
Previous work has demonstrated that virtualization side channels can be used to measure co-resident server load [37]. We build on this work with co-resident watermarking, discovering more accurate traffic information about our target's business. We accomplish this by simply monitoring the throughput 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 TCP 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, both flows' throughput will decrease equally and the ratio will remain constant. Changes in the load of other customers' virtual machines also affect both the CLIENT and FLOODER equally, and therefore the ratio will be maintained. The only scenario in which the ratio changes 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, then performed rapid measurements on both flows. Next, different load events were introduced and observed. For the first trial, pictured in 8b, an increasing number of web requests were issued from another host on the local network in ten-second intervals. The CLIENT calculated exponentially weighted moving averages of the two flows' packet arrivals, 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 8c, web requests are instead issued to other co-resident virtual machines. Every 22.5 seconds, 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.

8. DISCUSSION

Co-resident network flows represent a versatile side channel inside the cloud. 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. Secondly, 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.



(a) A decodable side channel message (b) Analysis: SERVER load increases. (c) Analysis: cloud node load increases.

Figure 8: Network flow side channel applications

8.1 Invisibility

In this work we do not focus on creating an invisible watermarking scheme. Currently, the FLOODER's activity would arouse immediate suspicion from any administrator. While invisibility is a desirable property of a watermark, recent work has demonstrated that it is extremely difficult to achieve [8, 17, 29]. However, in co-resident watermarking the attacker has the built-in advantage of being expected to create some reasonable amount of delay for his fellow customers. By creating a more realistic traffic model for the FLOODER, we believe it would be possible to perform co-resident watermarking without announcing the presence of malicious activity. This would of course come at the cost of the scheme's performance, as flooding intervals would need to occur less frequently.

8.2 Defenses

There are several defenses against the attack we've proposed; however, all have serious drawbacks associated with them:

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. However, 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 saw that co-resident watermarking can be thwarted by net under provisioning of instances relative to the network transmission speed of their physical host. This made it difficult to launch our attack on third party clouds. Unfortunately, this defense also depends on wasting resources, which impacts the bottom line of cloud providers. Additionally, studies point to a rapid increase in VM density that makes a communications bottleneck more feasible [16].

3. Alternatively, cloud instance administrators may provision their networks to not take advantage of the "free" bandwidth that is available in a multiplexed environment. Again, this will negatively impact the relative value of using cloud-based service providers. While this could be seen as a defensive measure against malicious co-residents, it's worth noting that our attack doesn't violate major SLAs [1].

4. 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 manufacturers are much more interested in addressing virtualization's performance challenges than those of security. SR-IOV and other pass-through technologies increase the exposure of underlying hardware and increase the effectiveness of side channels.

5. Another possible avenue of defense would be to use the random scheduling mechanism previously employed in cache measurements [22] to do random outbound packet scheduling. While this would be effective on some level, it could trigger TCP congestion control [40] and degrade performance across all virtual machines. In this sense 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, this would break certain network related aspects of virtual machine scheduling by the hypervisor.

The problem we illustrate is inherent in resource sharing, and is particularly essential to cloud computing's economy that is based around maximizing resource utilization. By launching co-resident watermarking on 3 of the major virtual hypervisors, we have demonstrated the presence of systemic resource-sharing vulnerabilities that are not unique to a particular virtualization initiative. Moreover, we have demonstrated that this problem can be further exacerbated by delegating resource management to the hardware level. 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.

9. RELATED WORK

9.1 Cloud Side Channels

Bowers et al. have proposed use of a different network timing side channel in order to challenge fault tolerance guarantees in storage clouds [6]. 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.

Cache-based side channels exploit the timing difference in access latency's between the cache and main memory. In the context of cloud computing, cache-based side channel attacks have attracted the most attention. Ristenpart et al. [37] 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. [47] utilized cache-based side channels as a defensive mechanism. Their scheme works by keeping portions of cache silent and measuring whether it has been accessed by other instances. Leveraging this scheme, they can challenge correct functionality on the part of the cloud provider and discover other unanticipated instances sharing the same host.

9.2 Hypervisor Security

Raj et al. [35] 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 [23]. 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 [18].

Other work aims to combat virtualization vulnerabilities by reducing the role and size of the hypervisor. Most drastically, Keller et al. eliminate a large attack surface by proposing the near elimination of the hypervisor [22]. This is achieved through pre-allocation of resources, limited virtualized I/O devices, and modified guest operating systems. While this approach in arguably 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 [30, 39]. We do not consider these systems in our work because they are not intended for the third party compute cloud model.

9.3 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 [12]. 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 [13]. 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 [41]. 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 [14]. Lastly, in 2012 partial source code for VMWare's ESX hypervisor leaked [7], and while no exploits have been directly attributed to this leak yet, such incidents increase the risk of compromise.

10. 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 watermarking scheme can be used to make a determination of co-residency in under 10 seconds 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. These investigations further demonstrate the ramifications of multiplexing hardware in virtualized environments, and is the beginning of a line of inquiry into designing hardware for the cloud that is performant without introducing undesired side effects.

Acknowledgements

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.

11. 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] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, et al. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, University of California, Berkeley, 2009.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. 19th ACM Symp. on Operating Systems Principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[5] A. Blum, D. Song, and S. Venkataraman. Detection of interactive stepping stones: Algorithms and confidence bounds. *Proc. Recent Advances in Intrusion Detection (RAID)*, 2004.

[6] K. D. Bowers, M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest. How to Tell if Your Cloud Files Are Vulnerable to Drive Crashes. In *CCS '11: Proc. 18th ACM Conf. on Computer and Communications Security*, pages 501–514, Chicago, IL, USA, 2011.

[7] J. Brodkin. 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`.

[8] S. Cabuk, C. E. Brodley, and C. Shields. Ip covert timing channels: design and detection. In *Proc. 11th ACM conference on Computer and communications security*, CCS '04, pages 178–187, New York, NY, USA, 2004. ACM.

[9] S. Cabuk, C. E. Brodley, and C. Shields. IP Covert Channel Detection. *ACM Transactions on Information and System Security (TISSEC)*, 12(4), Apr. 2009.

[10] S. Chinni and R. Hiremane. Virtual Machine Device Queues. White paper, Intel Corporation, 2007.

[11] B. Coskun and N. Memon. Online sketching of network flows for real-time stepping-stone detection. In *Proc. 2009 Annual Computer Security Applications Conf.*, ACSAC '09, pages 473–483, Washington, DC, USA, 2009. IEEE Computer Society.

[12] 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`.

[13] CVE-2007-5497. Multiple integer overflows in libext2fs. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5497`.

[14] CVE-2010-2240. The do_anonymous_page function in mm/memory.c. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2240`.

[15] Y. Dong, Z. Yu, and G. Rose. SR-IOV Networking in Xen: Architecture, Design and Implementation. In *Proc. First Conf. on I/O Virtualization*, WIOV'08, page 10, Berkeley, CA, USA, 2008. USENIX Association.

[16] S. Gamage, A. Kangarlou, R. R. Kompella, and D. Xu. Opportunistic Flooding to Improve TCP Transmit Performance in Virtualized Clouds. In *Proc. 2nd ACM Symp. on Cloud Computing*, SOCC '11, pages 1–14, New York, NY, USA, 2011. ACM.

[17] S. Gianvecchio and H. Wang. Detecting covert timing channels: an entropy-based approach. In *Proc. 14th ACM conference on Computer and communications security (CCS'07)*, Alexandria, VA, USA, 2007.

[18] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing Performance Isolation Across Virtual Machines in Xen. In *In Middleware*, 2006.

[19] I. Habib. Virtualization with KVM. *Linux Journal*, Feb. 2008.

[20] A. Houmansadr and N. Borisov. SWIRL: A Scalable Watermark to Detect Correlated Network Flows. In *Proc. 18th ISOC Symp. on Network and Distributed Systems Security (NDSS '11)*, San Diego, CA, USA, Feb. 2011.

[21] A. Houmansadr, N. Kiyavash, and N. Borisov. RAINBOW: A Robust and Invisible Non-Blind Watermark for Network Flows. In *Proc. 16th Network and Distributed System Security Symp. (NDSS'09)*, February 2009.

[22] E. Keller, J. Szefer, J. Rexford, and R. B. Lee. Eliminating the Hypervisor Attack Surface for a More Secure Cloud. In *Proc. ACM Conf. on Computer and Communications Security (CCS'11)*, Oct. 2011.

[23] G. Keramidas, A. Antonopoulos, D. Serpanos, and S. Kaxiras. Non Deterministic Caches: A Simple and Effective Defense Against Side Channel Attacks. *Design Automation for Embedded Systems*, pages 221–230, 2008.

[24] N. Kiyavash, A. Houmansadr, and N. Borisov. Multi-flow Attacks Against Network Flow Watermarking Schemes. In *Proc. 17th USENIX Security Symp.*, San Jose, CA, 2008.

[25] P. Kutch. PCI-SIG SR-IOV Primer. Technical report, Intel Corporation, 2011.

[26] A. M. Law and D. W. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 2000.

[27] X. Luo, E. Chan, and R. Chang. Cloak: A Ten-Fold Way for Reliable Covert Communications. In *Proc. European Symp. on Research in Computer Security ESORICS*, 2007.

[28] X. Luo, J. Zhang, R. Perdisci, and W. Lee. On the Secrecy of Spread-Spectrum Flow Watermarks. In *Proc. European Symp. on Research in Computer Security ESORICS*. 2010.

[29] X. Luo, P. Zhou, J. Zhang, R. Perdisci, W. Lee, and R. K. C. Chang. Exposing Invisible Timing-based Traffic Watermarks with BACKLIT. In *Proc. 27th Ann. Comp. Sec. Applications Conf.*, ACSAC '11, Orlando, FL, USA, Dec. 2011.

[30] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *Proc. 2010 IEEE Symp. on Security and Privacy*, Oakland, CA, USA, May 2010.

[31] S. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *Proc. 2005 IEEE Symp. on Security and Privacy*, Oakland, CA, USA, May 2005.

[32] K. Okamura and Y. Oyama. Load-based covert channels between Xen virtual machines. In *Proc. 2010 ACM Symp. on Applied Computing*, SAC '10, Sierre, Switzerland, 2010.

[33] P. Peng, P. Ning, and D. S. Reeves. On the Secrecy of Timing-Based Active Watermarking Trace-Back Techniques. In *Proc. 2006 IEEE Symp. on Security and Privacy*, Oakland, CA, USA, 2006.

[34] A. N. Pettitt and M. A. Stephens. The Kolmogorov-Smirnov Goodness-of-Fit Statistic with Discrete and Grouped Data. *Technometrics*, 19(2):205 – 210, 1977.

[35] H. Raj, R. Nathuji, A. Singh, and P. England. Resource Management for Isolation Enhanced Cloud Services. In *Proc. 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, Chicago, IL, USA, 2009.

[36] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, A. L. Cox, and S. Rixner. Achieving 10 Gb/s using Xen Para-virtualized Network Drivers. Xen Summit, Febuary 2009.

[37] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *CCS'09: Proc. 16th ACM Conf. on Computer and Communications Security*, Chicago, IL, USA, October 2009.

[38] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance. *Proc. VLDB Endowment*, 3(1-2):460–471, Sept. 2010.

[39] A. Seshadri, M. Luk, N. Qu, and A. Perrig. SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes. In *SOSP'07: Proc. 21st ACM Symp. on Operating Systems Principles*, Stevenson, WA, USA, 2007.

[40] W. R. Stevens. *TCP/IP illustrated (vol. 1): the protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[41] 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`.

[42] X. Wang, S. Chen, and S. Jajodia. Network Flow Watermarking Attack on Low-Latency Anonymous Communication Systems. In *Proc. 2007 IEEE Symp. on Security and Privacy*, Oakland, CA, USA, May 2007.

[43] X. Wang and D. S. Reeves. Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Interpacket Delays. In *Proc. 10th ACM conference on Computer and communications security*, CCS '03, pages 20–29, New York, NY, USA, 2003. ACM.

[44] J. Whiteaker, F. Schneider, and R. Teixeira. Explaining Packet Delays Under Virtualization. *SIGCOMM Computer and Communication Review*, pages 38–44, 2011.

[45] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. An Exploration of L2 Cache Covert Channels in Virtualized Environments. In *Proc. 3rd ACM Workshop on Cloud Computing Security (CCSW'11)*, Nov. 2011.

[46] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao. DSSS-Based Flow Marking Technique for Invisible

Traceback. In *Proc. 2007 IEEE Symp. on Security and Privacy*, May 2007.

[47] Y. Zhang, A. Juels, A. Oprea, and M. Reiter. HomeAlone: Co-Residency Detection in the Cloud via Side-Channel Analysis. In *Proc. 2011 IEEE Symp. on Security and Privacy*, Berkeley, CA, USA, May 2011.

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 provide 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 dom0 corresponds 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 packet domU packet passes through dom0 on its way to or from the physical interface. Xen packet scheduling is 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 100ms to round-trip times [44], limiting network throughput to as little as 2.9 Gbps [36]. 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 add 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 co-scheduling 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 re-use 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)[19]. 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, 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.

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 10GB 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) [10] and Single Root I/O Virtualization (SR-IOV) [15].

VMDq is 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.