An Exploration of L2 Cache Covert Channels in Virtualized Environments

Yunjing Xu, Michael Bailey, Farnam Jahanian
Dept. of Computer Science and Engineering, University of Michigan
Email: {yunjing, mibailey, farnam}@eecs.umich.edu

Kaustubh Joshi, Matti Hiltunen, Richard Schlichting
AT&T Labs Research
Email: {kaustubh, hiltunen, rick}@research.att.com

ABSTRACT

Recent exploration into the *unique* security challenges of cloud computing have shown that when virtual machines belonging to different customers share the same physical machine, new forms of cross-VM covert channel communication arise. In this paper, we explore one of these threats, L2 cache covert channels, and demonstrate the limits of these this threat by providing a quantification of the channel bit rates and an assessment of its ability to do harm. Through progressively refining models of cross-VM covert channels from the derived maximums, to implementable channels in the lab, and finally in Amazon EC2 itself we show how a variety of factors impact our ability to create effective channels. While we demonstrate a covert channel with considerably higher bit rate than previously reported, we assess that even at such improved rates, the harm of data exfiltration from these channels is still limited to the sharing of small, if important, secrets such as private keys.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Invasive software (e.g., viruses, worms, Trojan horses)*

General Terms

Experimentation, Measurement, Security

Keywords

covert channel, bit rate

1. INTRODUCTION

Major public cloud service providers such as Amazon (EC2) [4] and Rackspace [30] have realized a vision of cloud computing that provides "the illusion of infinite computing resources available on demand" and "the elimination of an up-front commitment" [5]. The economic value of this paradigm is clear: the cloud computing market place will grow substantially by generating revenues of $68.3 billion in 2010 and reaching $148.8 billion by 2014, from $58.6 billion in 2009 [18]. While compelling, the cloud computing paradigm still faces several daunting challenges, most notably, assuring the security and availability of the services it provides.

Recently, attention has been drawn to the security of cloud computing with a particular focus on determining what is novel (e.g., covert and side channels, longer trust chains, other cloud customers could be subverters, etc.), and what is not (e.g., phishing, data loss, botnets, etc.) [11]. Of these, the implicit mutual trust between cloud customers has received perhaps the most consideration. As the cloud hardware is shared, potentially between different customers, the opportunity of building *new covert channels unique to the cloud computing environment* arises. Given the economic impact of cloud computing, it is not surprising that researchers have already proved the feasibility of this type of threat [27, 31].

Perhaps the most notable of these demonstrations is the seminal work of Ristenpart et al. [31] in which the authors demonstrate a cross-VM covert channel is possible in Amazon's EC2. This two step attack first requires the attacker to place his own VM on the same physical machine as the target VM. Once the malicious VM and its target VM are co-located, the second step is to extract confidential information from the target VM by abusing the shared hardware. Using a technique for encoding information into the access latencies of a shared L2 cache [29], Ristenpart et al. [31] showed that confidential information can be leaked from the target VM to the malicious VM.

Such demonstrations are interesting not because public clouds, such as Amazon's EC2, place restrictions on information flow (they do not), but rather because it is easy to envision cloud environments in which information flow controls (IFCs) are important. For example, the recent rapid expansion of cloud-based projects from such sensitive agencies as Defense Information Systems Agency (DISA) and Department of Energy (DOE) [12] underscore the need to investigate novel data ex-filtration techniques and defenses. Further, what is compelling about these demonstrations is not that they represent the best way to exfiltrate information (for example, one could build a network-based covert channel [9], or encode data in responses to public queries), but rather that such a channel is *unique to the cloud computing environment.*

This idea has become so ingrained the cloud security cul-

CCSW'11, October 21, 2011, Chicago, Illinois, USA.

ture, that Ristenpart et al. [31] is often cited, and covert channels in particular noted, as one of core challenges in cloud computing security. For example, in examining new opportunities for unique cloud computing security research, Chen et el. [11] suggest that: "Side channels and covert channels pose another fundamental threat ... While not a panacea (e.g., it takes very few bits to steal a password), a helpful analysis could include when appropriate a quantification of channel bit rates, coupled with an assessment of the bit rate required to do harm." Others too have reflected on cloud security and noted: "another issue to consider in isolating processing on the same physical machine is whether an attacker can use covert channels to leak sensitive information...at present, no covert channels are known to be effective in cloud systems." [23].

Our goal in this paper is to expand the scope of this pioneering work into the novel threats facing the cloud computing environment by directly addressing the need for quantification of the bit rates of L2 cache, cross-vm covert channels and assessment of their ability to do harm. We note that such an analysis was never a specific goal of Ristenpart et al., but rather they show, importantly, that "Covert channels provide evidence that exploitable side channels may exist" [31]. In an effort to quantify the reasonable limits of these channels, we begin with idealized models of the channel behavior, refining that model as practical aspects of such channels, most notably the configuration of resource sharing in EC2, mandate changes. As a result we are able to not only verify the previous experimental results, but also provide an in-depth analysis of the mechanisms involved and their effects. Given that "no attempts were made at optimizing the bandwidth of the covert channel" [31], it is perhaps unsurprising that the previously proposed covert channels provided a bit rate of 0.2 bps, a mere fraction above the government minimum standard of 0.1 bps [1]. However, through our detailed analysis, we arrive at a low error rate, covert channel with considerably higher bandwidth than those previously reported. With respect to our assessment of the ability to do harm, we still find such bit rates limited to the exfiltrations of small secrets such as passwords and keys.

2. BACKGROUND

2.1 The Xen Hypervisor

The Xen hypervisor is an open source virtualization solution for various platforms including x86, x86_64, IA64, and ARM [7]. It supports a virtualization technique called *paravirtualization*, in addition to full virtualization. Compared to full virtualization, paravirtualization does not need any hardware support so that it can be used on legacy hardware, while still providing acceptable performance. On the other hand, paravirtualization requires modification to the kernel of guest operating systems to directly interact with the underlying hypervisor via *hypercalls*, which are analogous to syscalls in the operating system. The Xen hypervisor only provides functionality such as resource protection and scheduling, and the hypercall interface. Each guest operating system running on top of Xen is called a *domain*. Virtual machine management and device drivers are delegated to a special privileged domain called *dom0*. Other non-privileged domains known as *domU* can only access devices indirectly via dom0.

Each Xen domain can be allocated with one or more virtual CPUs (VCPUs). The scheduling of VCPUs on physical CPUs is handled by Xen's credit scheduler [34]. By default, the credit scheduler allocates each VCPU certain credits worth 30ms of CPU time so that each VCPU can occupy a physical CPU core up to 30ms before being de-scheduled. The scheduler ticks every 10ms to charge the current VCPU for the physical CPU time it used. If that VCPU has no credit left, it will be de-scheduled and assigned a priority called *under*, which means before the next credit allocation it is only allowed to run when no other VCPU has credits left. The VCPUs with credits left have a higher priority called *over* so that they will be favored when the scheduler wants to pick the next VCPU to run.

2.2 Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (EC2) [4] is a public cloud service with an infrastructure-as-a-service model. It allows developers to rent virtual machine instances from its data center in a pay-as-you-go manner. EC2 uses a customized version of Xen to support the service. We are not aware of the exact algorithm used to schedule VCPUs in EC2, the credit scheduler comes as a default in Xen is thus assumed in the following discussion. The types of instances available include micro, small, large, and extra large, and each has different capabilities for CPU, memory and I/O. Among all the instance types, a small instance is allocated a single VCPU with a fixed 40% cap. In addition, multiple small instances share a single physical machine. Thus, small EC2 instances are used in our experiments to evaluate L2 cache-based covert channels.

2.3 Cache-Based Covert Channels

Ristenpart et al. first introduced the concept of cross-vm covert channels [31]. Their basic idea is to construct certain patterns of contention on the hardware resources shared by two co-located VMs and use the contention patterns to encode information. For example, to send a single bit via a shared hard disk, attackers may let both the sender and the receiver VMs operate on large files concurrently for a specific period of time. During that time, the sender can choose to read files or do nothing to represent bit one or zero. At the same time, the receiver can distinguish the two by timing its own disk operations to decode the information.

Based on the above idea, Ristenpart et al. implemented three proof-of-concept covert channels on EC2: a 0.006bps channel using memory bus contention, and 0.0005bps channel using hard disk contention, and a 0.2bps channel using L2 cache contention [31].

While the number of possible channels is open-ended, we have opted to focus on *L2 cache covert channels* as described in prior work [31] for a variety of reasons: (i) it arguably has the highest potential bit rate as the time needed to make a contention measurement with modern L2 cache is on the scale of milliseconds, but the reported rate in prior work was very small, (ii) prior work only discussed such channels in brief and did not provide an in-depth analysis of the mechanisms involved and their effects, both qualitative and quantitative, in various settings, (iii) reproducing these results bolsters confidence in previous experimental results and allows for an exploration of changes to EC2 allocation and placement strategies since the initial work.

In particular, the L2 cache-based information encoding scheme proposed by Ristenpart et al. can be summarized as

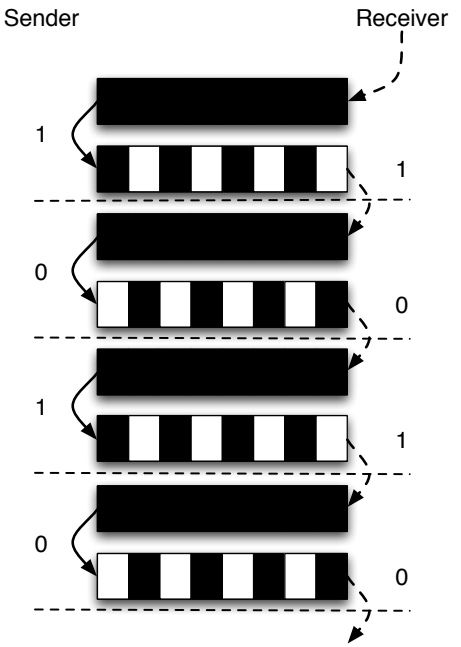

Figure 1: An illustration of a covert channel using L2 cache to encode information. For each bit, the sender evicts half of the cache lines from the L2 cache saturated previously by the receiver (solid lines). The receiver then decodes the information by measuring the difference of timing in accessing different subsets of the cache (dashed lines).

follows [31]: All the cache lines are divided into two subsets (a and b). To send a bit, the sender evicts the receiver's cache content from the cache lines correspond to one subset and leave the other untouched by accessing the memory addresses mapped to the chosen cache lines. Then, the receiver can decode the information by comparing the timing in accessing the two subsets separately, and if subset a takes significantly longer to read than subset b, it is bit one; otherwise it is bit zero. This process is illustrated by Figure 1.

3. ASSESSMENT OF THE BIT RATE REQUIRED TO DO HARM

Obviously, a covert channel with the bit rate of 0.2bps cannot be leveraged to exfiltrate large data records. For example, let's look at the numbers from the Digital Forensics Association. From 2005 to 2009, the number of lost records in data breaches ranged from 1 to 130,000,000 with an average of 387,926 [16]. While the report did not indicate the size of a lost record, we can explore the variety of record types to understand the practical impact of the channel bit rate. For example, assume that each record contains the information from the first track of a credit card magnetic strip [33] (i.e., name, number, dates, etc.). If we use a 32-bit encoding scheme (while more effective encoding schemes may exist, we simply choose the easy to implement UTF-32), the size of a single record is 2,528 bits. It would take 3.51 hours to leak this piece of information with a 0.2bps covert channel. Transferring all the information contained in a average data breach would take more than a century at 0.2bps bit rate. If the records leaked are medical data, each single record may contain 1,024,000 bits [8]. In this case, even leaking a single record would take the 0.2bps channel as long as 1422.22 hours.

Therefore, covert channels with this scale of bit rate may only be useful in leaking small cryptographic secrets. For example, the 2048-bit private key owned by the author contains 1,743 bytes. It will take about 20 hours to leak this key with the 0.2 bps L2 cache covert channel. However, if using a channel with the maximum bit rate 262.47bps, as described in § 4, leaking the same private would only take about 53 seconds. It should be noted that these are simply illustrations and one does not always need to leak an entire secret key to compromise it [14]. Often very few bits suffice to reconstruct a cryptographic secret and in such cases even low bandwidth covert channels are important.

Given the 1000x difference between the bit rate achieved in existing work on EC2 [31] and the maximum bit rate we calculated, we find ourselves compelled to ask:

> "What is the maximum achievable bit rate in practice over cross-VM covert channels? What factors influence the bit rate achievable in cross-VM covert channels?"

4. NAIVE QUANTIFICATION OF CHANNEL BIT RATES

Before discussing the design and implementation of the L2 cache covert channels, we present a back-of-the-envelope calculation for its maximum bit rate. The numbers used for the discussion in this section are derived with the following L2 cache specifications:

- L2 cache size: 6MB
- L2 cache line size: 64B
- L2 cache associativity: 24-way

In addition, we use *7 nanoseconds* as the time to fetch from the L2 cache, and *100 nanoseconds* as the time to fetch from main memory [15]. With these numbers, estimating the time to transmit a symbol on the covert channel is straightforward. The number of bits that can be encoded into a symbol is an open question and depending on the CPU design, it may range from one to $log(\#$ of cache lines). In order to compare with existing work [31], we assume that each symbol contains one bit of information. Thus, we can use the same information encoding scheme described in the background section.

In the ideal case, the minimum time to send a bit using L2 cache is the sender's write time (T_w) plus the receiver's read time (T_r). Thus, the maximum bit rate would be $1/(T_w+T_r)$. And this ideal case can be summarized using Protocol 1($P1$). While this initial protocol is very simple, it will evolve as the channel environment becomes more and more complicated.

Now we just need to estimate T_w and T_r to calculate the maximum bit rate. However, considering the fact that L2 cache is usually shared by both code and data, these two numbers become difficult to compute. In addition, modern CPU features such as prefetching will further complicate this task. However, we ignore these complications for the

Protocol 1 ($P1$)

1: The sender and receiver both allocate a buffer with the same size as the L2 cache. Then the receiver accesses its own buffer to fulfill the cache lines (one time initialization).
2: Repeat Step 3 and 4 sequentially for each bit.
3: The sender accesses subset a (or b) to send bit one (or zero) (T_w).
4: The receiver accesses subset a and b separately to decode the information (T_r).

moment, and assume T_w and T_r can be naively estimated as follows:

$$\begin{aligned} T_w &= 6MB/64B/2 * 100ns \approx 5ms \\ T_r &= 6MB/64B/2 * (100ns + 7ns) \\ &\approx 5ms + 0.34ms = 5.34ms \end{aligned}$$

Here the cache size is divided by the cache line size because, in order to fill up a cache line, it only needs to be accessed once by any byte within it. As a result, the maximum bit rate using the L2 cache in this idealized case would be

$$1bit/(5ms + 5.34ms) \approx 96.71bps$$

If more realistic numbers are desired, the complications discussed above must be put back into the calculation. To solve this problem, instead of looking into the hardware design manual to refine the estimation, we simply implement the basic operations of Step 3 & 4 in $P1$ on our laboratory machines, which has the same L2 cache configuration and 2.83GHz clock speed, and then profile T_w and T_r by executing these operations. The results of profiling for T_w and T_r are 1.47ms and 2.34ms respectively. Therefore, the maximum bit rate of L2 cache covert channel on our laboratory machines would be

$$1bit/(1.47ms + 2.34ms) \approx 262.47bps$$

5. ACHIEVABLE BIT RATES IN THE LABORATORY

The channel bit rate estimated with protocol $P1$ is unrealistic and, in particular, the idealized model overlooks the following factors:

- The operations of the sender and the receiver cannot be synchronized perfectly.
- There is other overhead associated with the channel program, such as process creation and destruction, that reduces the channel bit rate.

Specifically, the sender and receiver, as two separate VMs, have no method to synchronize perfectly in a way that the receiver's read can follow immediately after the sender's write. To solve this problem, Ristenpart et al. used a busy loop at the receiver's side to wait for the sender until the receiver perceives a large jump on its CPU counter [31]. Clearly, this solution implies two requirements: a) the two VMs share the same core at least for the next bit ($R1$), b) their VCPUs are pinned to the shared core ($R2$), otherwise the receiver would not be able to notice the timing of the sender's operations.

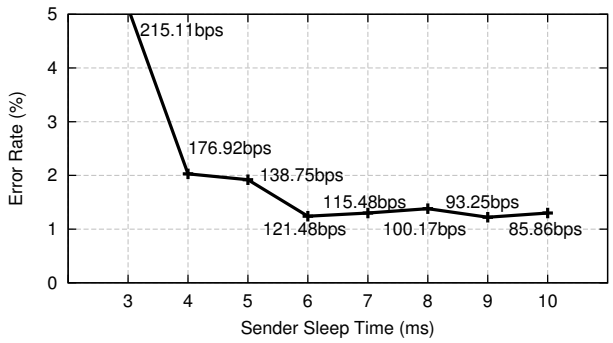

Figure 2: The error rate for the covert channel built in laboratory environment drops as the sleep time increases. When the sleep time is larger than or equal to 6ms, the error rate becomes stable.

Now we derive a new Protocol 2 ($P2$) that is applicable in a laboratory environment.

Protocol 2 ($P2$)

1: One time initialization. Same as $P1$
2: Repeat Step 3 and 4 sequentially for each bit.
3: The sender accesses subset a (or b) to send bit one (or zero) (T_w). Then it goes to sleep for T_s.
4: The receiver busy loops until CPU counter jumps by at least N_j. Then it accesses subset a and b separately to decode the information (T_r).

For the second problem, the other overhead can only be estimated by actually running the covert channel on lab machines. To implement this new protocol and estimate its potential bit rate, we just need to determine the value of T_s and N_j. The minimum value required for T_s should be large enough to allow the receiver to finish its operation, i.e., $T_s \geq T_r = 2.34ms$. Consider the variability of T_r, we conservatively choose $T_s = 3ms$ as the minimum sleep time. In addition, N_j should be large enough to cover the cycles needed to for the sender to finish its cache operation. Given a CPU with 2.83GHz clock speed, $N_j \geq 1.47 * 2830000 = 4160100$. Consequently, a practical bit rate that is possible to be realized in a laboratory environment that satisfies requirements $R1$ and $R2$ would be:

$$1bit/(1.47ms + 3ms) \approx 223.71bps$$

The difference between 223.71bps and the theoretic maximum 262.47bps can be considered as the *synchronization overhead* in practice. We implement this channel using protocol $P2$ on our lab machines by transmitting a 64-byte text 100 times. Using the above parameters, a bit rate of 215.11bps is obtained with an error of 5.12% on average. The extra loss in bit rate is due to the *overhead of non-cache-related code*, such as process creation/destroy, and the code that coverts the source of information into raw bits for the sender.

In order to reduce the error rate for the channel, we can increase the value of T_s because the uncertainties generated by the process/VM scheduling algorithms and other environmental factors may cause the channel out-of-sync occasionally. Figure 2 depicts the relationship between sender

sleep time and channel error rate. As we increase T_s, the resulting error rate drops. When the sleep time is larger than or equal to 6ms, the error rate becomes stable. As expected, when the sleep time T_s of the sender approaches its minimal, out-of-sync errors are more likely to happen. On the other hand, the impact of sleep time to the channel error rate diminishes when T_s become just large enough to tolerate the variability of T_r.

Before we close the discussion about channel bit rate in the laboratory environment, there is still one difference to be noted between $P2$ and the protocol proposed by Ristenpart et al.: the receiver also sleeps briefly every iteration in their version [31]. According to Ristenpart et al., the rationale behind the sleep is for the receiver "to build up credit with Xen's Scheduler" [31] because its busy loop may eat up scheduling credit quickly so that the receiver is less likely to be scheduled on time when there is a third VM (or more) sharing the same core. To give a complete picture, we define this version as Protocol 3 ($P3$).

Protocol 3 ($P3$)

1: One time initialization. Same as $P1$
2: Repeat Step 3 and 4 sequentially for each bit.
3: The sender accesses subset a (or b) to send bit one (or zero) (T_w). Then it goes to sleep for T_{ws}.
4: The receiver sleeps for T_{rs}, after which it busy loops until its CPU counter jumps by at least N_j. Then it accesses subset a and b separately to decode the information (T_r).

In our previous effort in building this covert channel, we started with this version of the protocol. And in the same laboratory environment, we only obtained a bit rate of around 30bps when $T_{ws} = 10ms$(or $20ms$), $T_{rs} = 3ms$ (both choices of T_{ws} produce a similar maximum bit rate), while the expected bit rate should be close to:

$$1bit/(1.47ms + 10ms) \approx 87.18bps$$
$$1bit/(1.47ms + 20ms) \approx 46.58bps$$

based on the above analysis. To explain this huge difference, we first look at the time to transfer one bit for 30bps case: $1s/30bps \approx 33.3ms$. Not surprisingly, this value is close to the maximum time slice that Xen's credit scheduler allows a virtual CPU to occupy a physical CPU [34]. Thus, the explanation for the anomaly is that when both the sender and receiver are put into sleep for each bit, the receiver always wakes up first to start its busy loop, during which the sender comes back to be runnable. If at this moment the sender preempts the receiver to running immediately, everything can stay as expected. However, in reality, the receiver may have accumulated enough credit that gives the receiver an equal priority (*over*) as the sender. Therefore, the sender may not be able to preempt the receiver, but has to be blocked for another 30ms in the worst case. While this scenario does not happen every iteration, it indeed happens frequently enough to reduce channel bit rate to be around 30bps despite the value of T_{ws} being 10ms or 20ms. And this phenomenon persists with protocol $P3$ until $T_{ws} > 30ms$, which effectively reduces the expected bit rate to be $\leq 30bps$. Consequently, in practice we suggest not to use protocol $P3$ unless a workload can justify its usage with improved error rate.

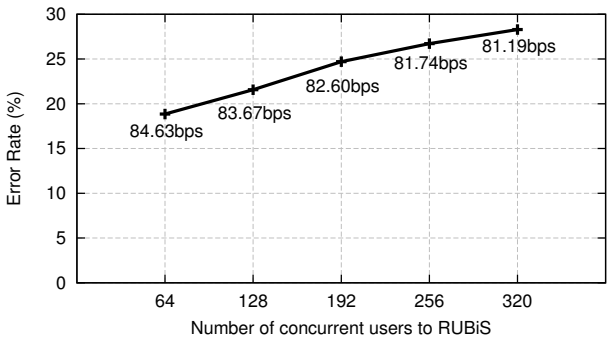

Figure 3: In the laboratory environments, if running a web application on a third VM sharing the same core as the covert channel, the channel error rate would increase and bit rate would decrease slowly as the number of users to that application grows.

6. ACHIEVABLE BIT RATES ON EC2

According to the above analysis and experiment results, the difference in bit rate between the derived maximum and the one reported in previous work [31] is more than three orders of magnitude. Given that we have not intentionally optimized our code either, we speculate that the following factors may contribute to this huge gap:

- Hardware specification
- Workloads in other VMs on the same physical host
- Hypervisor configuration, i.e., scheduling policies
- Design of the protocol

In this section, we analyze those factors in order and verify our speculation by implementing the L2 cache covert channel on EC2.

Hardware Specification. The small EC2 instances we use have the same L2 cache specification as our lab machines. However, the CPUs on our lab machines have a higher clock speed of 2.83GHz than that of 2.66GHz on the EC2 instances we use. Even if considering that the CPUs on EC2 may be one or two generations older, the impact of hardware specification on channel bit rate still should not be more than 10%. Moreover, one may notice that EC2 uses 40% CPU cap on small instances, which effectively reduces the clock speed by 60%. We discuss this problem in more detail when it comes to the hypervisor configuration.

Workloads. Given the design of the protocols $P1$ to $P3$ that the sender does not verify if the receiver has received the current bit and whether the received bit is correct, workloads on other VMs sharing the same hardware would only increase the error rate of the channel. Meanwhile, if the VMs with the workloads also share the same core as the covert channel VMs, then they will reduce the channel bit rate because they steal CPU cycles from the covert channel.

To support this argument, we introduce a third VM on our lab machines to share the same core with the other two VMs used for evaluating $P2$. On the third VM, we deploy a web application called RUBiS, which is an eBay-like online

auction application that is used extensively as a benchmark in the research community [10, 28]. In the default workload of RUBiS, it has a parameter called "number of clients per node" that controls the number of concurrent users accessing the application. We vary this parameter and run a covert channel in parallel on the other two VMs using $P2$ with 10ms sleep time. The result of error rate and bit rate are plotted in Figure 3. As the figure suggests, the error rate increases slowly as the number of concurrent users grows. At the same time, because more concurrent users means more cycles needed to process their requests, the channel bit rates also slowly decreases. However, compared to the original bit rate 85.86bps without third-party workload, the biggest drop in bit rate is no more than 6%. Thus, we speculate that the impact of workloads themselves to the covert channels on EC2 is very limited.

Hypervisor Configuration. Suppose the Xen hypervisor on EC2 has the same credit scheduler with identical parameters as the standard Xen scheduler, when protocol $P3$ is applied, the bit rate can be reduced by an order of magnitude. In addition, if we examine the CPU usage in small instances on EC2, a 40% cap should be noticed. And this cap may effectively reduce the channel bit rate by at least 60% over an extended period of time. Up to this point, the raw bit rate should have been reduced to about 10bps. Furthermore, the VCPUs used by EC2's small instances are not pinned to any specific physical cores. Using the CPUID instruction, which does not seem to be virtualized on EC2, small instances suggest four cores on a single physical machine, and VCPUs constantly migrate to different physical cores on the scale of milliseconds to thousands of milliseconds. It means at any moment, two VMs co-located on the same physical machine may not be able to communicate via the L2 cache simply because they are not running on the right cores. This policy together with the 40% cap violate both $R1$ and $R2$, which are required to ensure the high bit rate for the laboratory experiments. And they significantly affect the design of the protocol to be used on EC2.

Protocol Design. Ristenpart et al. use multiple samples for a single bit to compensate the negative impact of core migration [31]. Thus, the resulting bit rate depends on the number of samples taken per bit, given protocol $P3$. To continue the above rough estimation, it is reasonable to obtain a bit rate of 0.2bps should we use 50 samples per bit.

To sum up, the major factors that may significantly affect the channel bit rate on EC2 include Xen's scheduling algorithm, 40% CPU cap, non-pinned VCPUs (core migration), all of which are environmental factors, and the design of the communication protocol, which is an artifact of the environmental factors. In the follow subsection, we describe our experiments as concrete examples to explain how does the environmental factors reduce the channel bit rate by at least 2 orders of magnitude in practice on EC2 with our refined protocol.

6.1 EC2 Co-location Revisited

Ristenpart et al.'s work explored the relationship between EC2's instance placement and its network configuration and designed a method to co-locate the attacker's VM to its victim with a probability better than a brute-force approach [31]. In our case, we care about the maximum achievable channel bit rate on EC2, so only two VMs both controlled by an attack need to be co-located, And it is a much easier task than to co-locate with a targeted victim VM. In this subsection, we start with the experience learned from the existing work [31], then share our new or revised experience for achieving and verifying VM co-location in the case the Amazon has changed its polices and algorithms in response to that paper.

To begin with, because it is believed that VMs belong to the same account will never be placed on the same physical machine on EC2 [31], we start with two accounts, each of which allows 20 concurrently running VMs by default. Because of the "parallel placement locality" [31], which says VMs launched simultaneously at the same availability zone (which roughly corresponds to a data center) are more likely to be co-located, we launch 20 small instances for each account roughly at the same time. Then, the first hop IP (internal to EC2) of each VM's outbound route is checked. And if any pair from the two accounts share the same first hop, they are believed to be co-located [31] on the same physical machine.

Based on our recent experiments on the east region of EC2, two observations worth a discussion. First of all, for different accounts, the availability zones with the same name (e.g., us-east-1b) may not refer to the the same physical infrastructure. In other words, the mapping between the name of availability zones and their physical locations is different from account to account. Fortunately, the actual mapping seems to be fixed once an account is created, and the mapping can be extracted by combining the information provided by several standard EC2 commands [21]. Because we are not the first to discover this phenomenon or the matching technique, and this technique is also out of the scope of this paper, we encourage curious readers to read the referred blog post [21]. In addition, while for each trial we roughly get 5 VM pairs with the same first hop out of the 20 by 20 candidates, not all of them can be confirmed to be co-located by running the L2 covert channel to be described later in this section. Meanwhile, we find that the pairs of VMs that allow covert channel communication have a much lower round trip time (RTT) if we do a tcptraceroute against the VMs within the same pair than that for the pairs failing our covert channel (0.06ms vs. 0.2ms).

Interestingly enough, among all the confirmed co-located VM pairs, we find one pair that belong to the same account. This pair was created when EC2 was experiencing a major outage on its east region [17], and it never happened again after the outage was fixed. Therefore, we speculate that during the outage when not all the physical machines were usable, EC2 tried to fulfill as many custom requests at the cost of a reduced guarantee of fault tolerance.

6.2 L2 Cache-Based Covert Channel On EC2

Once the VM co-location is verified, we run our own version of the L2 cache-based covert channel on EC2 to explore its potential bit rate. As discussed above, all three protocols ($P1$ to $P3$) do not work well on EC2 due to the environmental factors related to the hypervisor configuration. As a response to these environmental factors, we present a refined protocol to explore and explain the practical bound on the bit rate of the L2 cache-based covert channel on EC2.

Recall that for protocol $P1$ to $P3$, a busy loop at the receiver's side is used for synchronization. However, given the

40% CPU cap alone, this method is guaranteed to lose information because blind spots will appear in the receiver's lifetime when the receiver VM is preempted by the hypervisor, and these blind spots can be large enough (60% of CPU time, hence on the scale of tens of milliseconds given the 10ms ticks of Xen's credit scheduler) to allow other VMs (or the sender itself) to overwrite the cache content before the receiver decodes any useful information. To solve this problem, we eliminate the synchronization mechanism entirely and simply busy loop with the cache operations at both sides of the channel. Here we define a *valid measurement of cache contention* as one side of the channel captures the time difference in accessing the cache subsets after the other side evicts certain cache lines. Then an invalid measurement would be that one side of the channel accesses the cache before the other side touches it, i.e., no time difference would be perceived. Consequently, the design of $P1$ to $P3$ implies that all the measurements taken during the channel lifetime are valid.

This design decision is made based on the fact that while when a valid measurement of cache contention will happen is unpredictable, once it happens the verification of the measurement is very accurate because the cache interference introduced by other processes or VMs are assumed to be uniformly distributed so that the difference between the time it takes to access the evicted cache subset and the non-evicted one remains noticeable (1.47 vs. 0.87ms in practice) on the scale of milliseconds.

However, this brute-force strategy leaves an obvious question: given that there is no guarantee for a valid measurement of cache contention to happen, even if you can take valid measurements once in a while, how do you verify whether the current bit has been transferred and when the sender and receiver should synchronously move on to the next bit. Even worse, the core migration on EC2 can make the cache contention between two given co-located VMs never happen in the worst case. Thus, the short answer to this question is that there can be *no* guarantee because we are trying to send information reliably on a unreliable channel, which is a problem similar to the TCP handshake protocol [22]. In fact, the protocols ($P1$ to $P3$) we have discussed so far are all one-way protocols, which means the receiver has no way to acknowledge the sender for the transmission, but it is not a problem when both sides can synchronize each other with the busy loop. Fortunately, if a valid measurement of cache contention happens to the receiver, it should have evicted the sender's content from the cache. As a result, the next time the sender takes a measurement, it will also have a chance to take a valid measurement. That means we can use a similar method to timing the cache operations at the sender's side as a pseudo-acknowledgement of the transmission. However, it suffers from the same problem that it has no guarantee to happen. To break this deadlock, similar to the scheme used by Ristenpart et al. [31], we simply repeat this procedure multiple times for each individual bit for transmission to increase the possibility of success. The above description is summarized as Protocol 4 ($P4$).

Before we move to the bit rate analysis with this protocol, the rationale behind Step 4 & 6 and the use of T_{ws} & T_{rs} should be explained first. Because contention measurements are now repeated multiple times for each bit, depending on the choice of N_w and N_r, either the sender or the receiver may get stuck in Step 3 and 5 respectively due to unexpected

Protocol 4 ($P4$)

1: One time initialization. Same as $P1$
2: Repeat Step 3-4 and 5-6 concurrently for each bit.
3: The sender accesses subset a (or b) to send bit one (or zero) (T_w). Repeat this process until obtains N_w valid measurements. After each measurement, despite valid or not, the sender sleeps for T_{ws}.
4: The sender accesses the cache subset opposite to the one used in Step 3. Repeat this process until an invalid measurement happens.
5: The receiver accesses subset a and b separately (T_r) to decode the information. Repeat this process until obtained N_r valid measurement. After each measurement, despite valid or not, the receiver sleeps for T_{rs}.
6: The receiver accesses the same subset used in Step 5. Repeat this process until an invalid measurement happens.

core migrations. As a result, Step 4 and 6 are used to check if the measurement loops used by the other side of the channel for the current bit have finished. For example, if Step 6 succeeds for the receiver, it means the sender is also likely to be working on Step 4, which implies they both can move on to the next bit. Again, there is still no guarantee that the success of Step 4 and 6 indicates that the sender and the receive have been synchronized for the next bit since core migration at that point can lead to the same perception, their presence increases the chance of being synchronized while their overhead on the bit rate is limited as the number of samples taken per bit increases. In addition, the sleep time of T_{ws} & T_{rs} are used to avoid repeating too fast to miss the opportunity of cache measurements, and 1ms will suffice. Please note that because for protocol $P4$ the sender and the receiver have a similar busy loop and also share the same sleep time, the sender's over-blocked problem caused by Xen's scheduler no long exists.

We implement $P4$ on EC2 with two co-located VMs found by the method described in the last subsection. To analyze the potential bit rate, we first profile the performance of basic cache operations to obtain T_w and T_r as follows:

$$\begin{cases} T_{w1} = 2.67ms, T_{r1} = 3.73ms & \text{if contention happens} \\ T_{w0} = 1.10ms, T_{r0} = 2.09ms & \text{otherwise} \end{cases}$$

Please note that the profiling result does not follow exactly the theory. For example, in the case of no contention happens, T_{r0} should be two times of T_{w0}. This slight miss match is just an artifact of a profiling based method itself, which is the best we can do on the black box system like EC2, and its impact to the analysis result should be limited. Meanwhile, the numbers may vary on a small scale if profiled with a different pair of co-located VMs.

In our implementation, the receiver is used to constantly monitor the cache content because while there is no guarantee on successful bit transmission, once it succeeds the chance for the receiver to get flipped bit is almost zero. And because the operations of the sender's and receiver's are always concurrent, we use the sender's side to analyze the channel bit rate in the following discussion.

In an ideal case, core migration never happens during the transmission so that every measurement will be valid. This overly optimistic assumption results in the following estima-

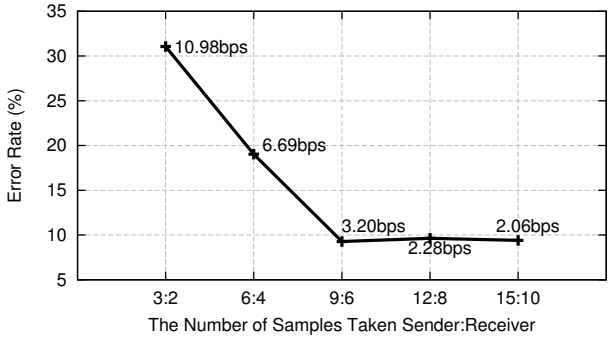

Figure 4: The error rate of the covert channel built on E2 using protocol $P4$ drops as more more samples $(N_w : N_r)$ are taken for each bit. After $N_w : N_r = 9 : 6$ no significant benefits can be gained by increase the number of samples

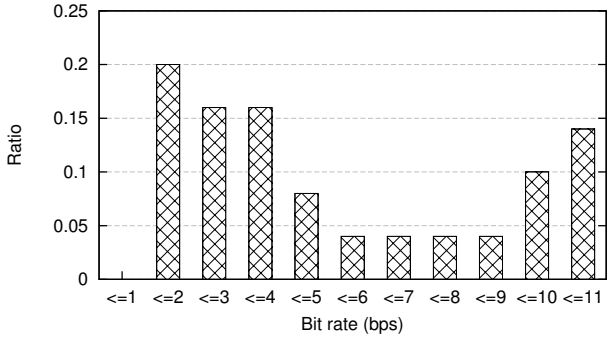

Figure 5: The bit rate distribution for the covert channel running on EC2 using protocol P4

tion of the ideal channel bit rate on EC2:

$$bitrate = 0.4 * 1bit/(N_w * (T_{w1} + T_{ws}) + T_{w0})$$

Again in the best case where we choose $N_w = 1$, the maximum bit rate would be

$$0.4 * 1bit/(2.67ms + 1ms + 1.10ms) \approx 83.86bps$$

However, this estimation is clearly unrealistic. Because it implies $N_r = 1$, but $T_{w1} \neq T_{r1}$ means N_w and N_r cannot take the same value. Instead, using $N_w : N_r \approx T_{r1} : T_{w1} \approx 3 : 2$ would be appropriate.

Intuitively, a small value of N_w tends to result in a high error rate because the sender and the receiver are more likely to be out-of-sync in this case. Figure 4 demonstrates the relationship between the error rates and the number samples $(N_w : N_r)$ with corresponding bit rate. While small values does give a higher error rate, once $N_w \geq 9$, the error rate becomes stable.

	Mean	Median	Max	Min
Bit Rate	3.20bps	3.75bps	10.46bps	1.27bps
Error Rate	9.28%	8.59%	28.13%	0%

Table 1: Basic statistics of the channel bit rate and error rate on EC2.

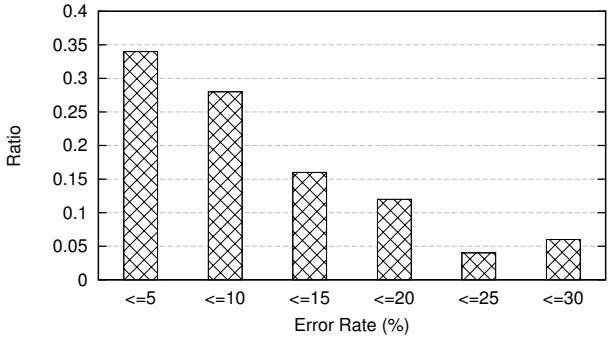

Figure 6: The error rate distribution for the covert channel running on EC2 using protocol P4

As result, we choose $N_w = 9, N_r = 6$ for further experiments. Figure 5 and Figure 6 shows the distribution of the bit rate and error rate for this pair of parameters. And Table 1 shows some basic statistics about these two metrics. The impact of the environment factors to these metrics is discussed in the following paragraphs.

First of all, given $N_w = 9$, if every cache measurement is valid, the expected channel bit rate would be

$$0.4 * 1bit/(9 * (2.67ms + 1ms) + 1.10ms) \approx 11.72bps$$

Considering other overhead associated with the covert channel program, the maximum bit rate 10.46bps that we obtain through experiment is very close to this expected number. Obviously, there two components that reduce the expected bit rate from 83.86bps to 11.72bps: the 40% CPU cap, which effectively reduce the bit rate by 60% if the channel runs for an extended period of time, and the repeated sampling, which is designed to deal with core migration and other uncertainties in VM scheduling.

Even worse, for the same reason, not every cache measurement can be valid. Thus, we need to obtain the average percentage of time that the sender spends on valid cache measurements among all the measurements that have been made during the transmission period. This number should be environment dependent and it can be only obtained through profiling. With the trials conducted on the same EC2 instances, we divide the cache measurements into three category using the timing of each measurement: valid measurements, invalid measurements, and measurements made when VM gets preempted. While the first two categories are self-evident, the last one contains the measurements that take more then 3ms to make, so that they are believed to be taken when corresponding VM is preempted by the hypervisor. The percentage of each category is listed as follows:

$$\begin{cases} 10.5\% & \text{Valid cache measurements} \\ 39.7\% & \text{Invalid cache measurements} \\ 6.5\% & \text{Measurements made when VM preempted} \end{cases}$$

As we can see, only 10.5% of the transmission time is actually spent on valid cache measurements. It explains why the experiments shows a bit rate of 3.20bps on average. These three categories make up 56.7% of the transmission time. The rest of it is in fact the 1ms sleep time after each measurement. As discussed before, this sleep time is used to avoid polling too fast for both sides. Its removal would significantly increase the error rate. On the other hand, reduce

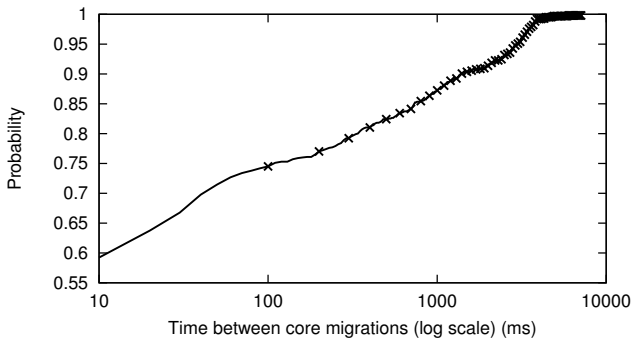

Figure 7: CDF of the time between core migrations. About 50% time a VCPU stays on the same core for no more than 10ms, while about 25% of time a VCPU stays on the same core for more than 100ms

the sleep time for each iteration would not improve the channel bit rate in a meaningful way because this large amount of sleep time is caused by the large percentage of invalid cache measurements, which in turn is caused by the uncertainties in VM scheduling such as core migration.

To understand why valid cache measurements only consist 10.5% of the transmission time, we profile the frequency of core migrations during our experiments. Please note that because to some extend a core migration is triggered by the execution of instructions, the profiling process itself would impact the profiled frequency. In addition, the profiling is conducted along side the covert channel communication, its resolution is therefore constrained by the polling frequency of the target channel. Consequently, the results should not be viewed as a characterization of the scheduling properties for the underlying scheduler, but just a concrete example to illustrate the impact of environment factors to the channel bit rate. Figure 7 shows the distribution of the time between core migrations. As we can see, almost 50% time a VCPU stays on the same core for no more than 10ms. This frequent migration will cause a large number invalid cache measurements. Meanwhile, about 25% of time, a VCPU stays on the same core for more than 100ms, which gives the channel plenty of time to make valid measurements. This perception also explains why the channel bit rate in Figure 5 shows a bimodal distribution.

7. DISCUSSION AND FUTURE WORK

§ 5 and § 6 together explain details about the maximum achievable bit rate in practice over L2 cache-based cross-VM covert channels in different environments with different constraints. Although the maximum bit rate we achieve on EC2 is over an order of magnitude faster then previously reported [31], the channel's capability to exfiltrate useful information is still limited, and it is only practical to leak cryptographic secrets. In future work we plan to explore threat models in which the leaking of these small secrets could lead to larger scale breaches.

In addition, we plan expand the work on EC2 to other comparable cloud service providers, such as Rackspace [30]. The goal then will be to explore the difference of environment factors for different providers and study their impact to the covert channel bit rate and error rate. For example, if a cloud provider does not put CPU cap on VCPUs but still allows core migration, can we significantly improve the channel bit rate? Based on these observations and measurements, we may further suggest industry best practice for containing covert channels.

Moreover, we plan to explore the natural open question of stronger L2 cache side channels attacks in virtualized environments.

8. RELATED WORK

In 1973, Lampson [26] discussed a classification of the ways in which information can be transferred between programs (i.e., legitimate, storage, and covert channels) and defined covert channels as "those not intended for information transfer at all, such as the service program's effect on the system load." Two classes of covert channels have emerged, those based on storage and those based on timing. In storage attacks, existing fields, memory locations, etc, are used to (secretly) encode information. For example, unused fields in network protocols can be used to convey information in a unintended way [2, 3, 20].

Timing-based covert channels are more sophisticated because the information is encoded by varying the timing of events in a system. Thus, the receiver of a timing-based covert channel must understand the original encoding scheme in order to obtain the actual information. For example, covert channels constructed using the time intervals between network packets fall into this category. Cabuk et al. designed such a covert channel using IP packets and proposed two statistical methods to look for the timing irregularity in the traffic as a detection mechanism [9]. In addition, Gianvecchio et al. designed an entropy-based solution to improve the detection accuracy [19].

Side channel attacks, however, exploit unintended system artifacts to learn about activities that are supposed to be secret. As such they are "in fact a covert channel without conspiracy or consent." [32] Leveraging cache-based side channels to extract cryptographic keys has been studied extensively [25, 29]. To mitigate the threat, researchers propose to inhibit the sources of timing channels. For example, Askarov et al. recently demonstrated that side channels could be mitigated to some extend by regulating all the timing events in the system [6]. In addition, Coppens et al. proposed to use a special compiler backend to eliminate timing behaviors [13]. Other solutions include totally disallowing the sharing of hardware in public clouds [24].

The subject of this paper, cross-VM covert channels, are a type of timing based attack. Ristenpart et al. [31] have proven its feasibility in a real cloud environment with an attack using shared L2 cache. As a follow up, Okamura et al. designed and evaluated an new attack that uses the load of a shared CPU to encode information [27]. On the other hand, Zhang et al. monitor the patterns of L2 cache usage within a guest domain to build a classifier of the usage patterns to check if there are other VMs sharing the same physical machine [35]. The main distinction of this work over the above mentioned work is a thorough examination of the factors that impact the bit rate of the L2 cache-based cross-VM covert channel in both a laboratory environment and a real world environment. It should be noted that side-channel attacks also make up a part of the threat posed by VM co-location, as discussed in Ristenpart et al. [31], but are beyond the scope of this work.

9. CONCLUSION

In this paper, we provide a quantification of the bit rates of a L2 cache-based channel and an assessment of its ability to do harm. We expand the scope of the pioneering work for this threat [31] by progressively refining models of cross-VM covert channels from the derived maximums, to implementable channels in the lab, and finally in Amazon EC2 itself. We show how a variety of factors impact our ability to create effective channels. While a covert channel with considerably larger bit rate than previously reported is demonstrated, we assess that even at such improved rates, the harm of data exfiltration from these channels is still limited to the sharing of small, if important, secrets such as private keys.

10. ACKNOWLEDGMENTS

This work was supported in part by the Department of Homeland Security (DHS) under contract number NBCHC080037, the National Science Foundation (NSF) under contract numbers CNS 1111699, CNS 091639, CNS 08311174, and CNS 0751116, and the Department of the Navy under contract N000.14-09-1-1042.

This material was based on work supported by the National Science Foundation, while working at the Foundation. Any opinion, finding, and conclusions or recommendations expressed in this material; are those of the author and do not necessarily reflect the views of the National Science Foundation.

11. REFERENCES

[1] Trusted computing system evaluation. "the orange book". dod 5200.28-std, U.S. Department of Defense., Washington, 1985.

[2] Abad, C. Ip checksum covert channels and selected hash collision. Technical report, 2010.

[3] Ahsan, K., and Kundur, D. Practical data hiding in tcp/ip. In *Proceedings of the 9th workshop on Multimedia & Security, (MM&Sec'02)* (Dallas, TX, USA, September 2002).

[4] Amazon. Amazon elastic compute cloud (ec2). `http://aws.amazon.com/ec2/`.

[5] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. *Commun. ACM 53* (April 2010), 50–58.

[6] Askarov, A., Zhang, D., and Myers, A. Predictive black-box mitigation of timing channels. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)* (Chicago, IL, USA, October 2010).

[7] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)* (Bolton Landing, NY, October 2003).

[8] Bowdena, K., MacCalluma, I., and Patience, S. A magnetic tape database for a real-time medical information system. *Computers in Biology and Medicine 5* (September 1975).

[9] Cabuk, S., Brodley, C. E., and Shields, C. Ip covert timing channels: Design and detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04)* (Washington, DC, USA, October 2004).

[10] Cecchet, E., Marguerite, J., and Zwaenepoel, W. Performance and scalability of ejb applications. In *Proceedings of the 17th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA'02)* (Seattle, WA, USA, November 2002).

[11] Chen, Y., Paxson, V., and Katz, R. H. What's new about cloud computing security? Tech. Rep. UCB/EECS-2010-5, EECS Department, University of California, Berkeley, Jan 2010.

[12] cloudbook: The Cloud Computing & SaaS Information Resource. Governemnt Cloud Computing Platforms. `http://www.cloudbook.net/directories/gov-clouds/government-cloud-computing.php`, 2011.

[13] Coppens, B., Verbauwhede, I., Sutter, B. D., and Bosschere, K. D. Practical mitigations for timing-based side-channel attacks on modern x86 processors. In *Proceedings of the 30th IEEE Symposium on Security & Privacy (S&P'09)* (Oakland, CA, USA, May 2009).

[14] Coppersmith, D. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *Journal of Cryptology 10* (1997), 233–260. 10.1007/s001459900030.

[15] Dean, J. Designs, lessons and advice from building large distributed systems. In *Proceedings of the 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware (LADIS'09)* (Big Sky, MT, USA, October 2009). Keynote speech.

[16] Digital Forensics Association. *The Leaking Vault Five Years of Data Breaches*, July 2010.

[17] EC2, A. Summary of the amazon ec2 and amazon rds service disruption in the us east region. `http://aws.amazon.com/message/65648/`.

[18] Gartner, Inc. Gartner Says Worldwide Cloud Services Market to Surpass $68 Billion in 2010. `http://www.gartner.com/it/page.jsp?id=1389313`, June 2010.

[19] Gianvecchio, S., and Wang, H. Detecting covert timing channels: An entropy-based approach. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07)* (Alexandria, VA, USA, October 2007).

[20] Giffin, J., Greenstadt, R., Litwack, P., and Tibbetts, R. Covert messaging through tcp timestamps. In *Proceedings of the 2nd Workshop on Privacy Enhancing Technologies (PET'02)* (San Francisco, CA, USA, April 2002).

[21] Hammond, E. Matching ec2 availability zones across aws accounts. `http://http://alestic.com/2009/07/\ec2-availability-zones`.

[22] Institute, I. S. RFC 793: Transmission control protocol, 1981. http://www.ietf.org/rfc/rfc793.txt.

[23] Jaeger, T., and Schiffman, J. Outlook: Cloudy with a chance of security challenges and

improvements. *IEEE Security and Privacy 8* (2010), 77–80.

[24] Keller, E., Szefer, J., Rexford, J., and Lee, R. B. Nohype: Virtualized cloud infrastructure without the virtualization. In *Proceedings of the 37th International Symposium on Computer Architecture (ISCA'10)* (Saint-Malo, France, June 2010).

[25] Kocher, P. C. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'96)* (London, UK, 1996).

[26] Lampson, B. W. A note on the confinement problem. *Commun. ACM 16* (October 1973), 613–615.

[27] Okamura, K., and Oyama, Y. Load-based covert channels between xen virtual machines. In *Proceedings of the 25th Symposium On Applied Computing (SAC'10)* (Sierre, Switzerland, March 2010).

[28] OW2. Rubis. `http://rubis.ow2.org/`.

[29] Percival, C. Cache missing for fun and profit. In *Proceedings of BSDCan 2005* (Ottawa, Canada, May 2005).

[30] Rackspace. Rackspace cloud hosting. `http://www.rackspacecloud.com/`.

[31] Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. Hey, you, get off of my cloud! exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)* (Chicago, IL, USA, November 2009).

[32] Tiri, K. Side-channel attack pitfalls. In *Proceedings of the 44th annual Design Automation Conference* (New York, NY, USA, 2007), DAC '07, ACM, pp. 15–20.

[33] Wiki. Magnetic stripe card. `http://en.wikipedia.org/wiki/Magnetic_stripe_card`.

[34] Xensource. Xen credit scheduler. `http://wiki.xensource.com/xenwiki/CreditScheduler`.

[35] Zhang, Y., Juels, A., Oprea, A., and Reiter, M. K. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy (S&P'11)* (Oakland, CA, USA, May 2011).