COMP0057 – Literature Review[⋆]
Establishing Cross-VM Communication Channels in IaaS Public Clouds

Nicolas Mohnblatt

UCL, `nicolas.mohnblatt19@ucl.ac.uk`

Abstract. In this survey, we consider the security of Infrastructure-as-a-Service clouds. Users of these services launch virtual machines (VMs) on shared hardware. We consider what could happen if an attacker registers as a legitimate user and attempts to break isolation boundaries between VMs. We find that researchers have managed to force the co-location of two attacker VMs, and force the co-location of an attacker's VM with that of a target. We investigate techniques that abuse the shared hardware to establish cross-VM communication channels. We find that state-of-the-art techniques allow to build covert channels robust enough to support a fully functional SSH connection.

Introduction

Infrastructure-as-a-Service (IaaS) is a business model for cloud computing that has grown in popularity in the last decade. In this model, a service provider offers its users to run *virtual machines* (VMs; also known as *instances* in the IaaS context) on the provider's hardware. By doing so, the provider can use the same hardware for multiple tenants, thus benefiting from economies of scale and reducing costs. To date, the three largest IaaS public clouds are Amazon Elastic Compute Cloud (EC2), Google Compute Engine (GCE) and Microsoft Azure.

Such a model may cause participants to share hardware with unexpected neighbours: one can imagine cases where a company's VM is running on the same physical machine as that of their direct competitor. When VMs run on the same physical machine, we say that these VMs are *co-resident* (or alternatively, *co-located*). Cases like the one described highlight the fact that isolation between co-resident VMs is a priority for IaaS cloud security. Typically, this isolation is provided in software through the use of virtualisation and a *hypervisor* – software that runs on a physical machine to schedule and switch between the execution of different VMs.

In this literature survey, we wish to explore the area of research that has focused on breaking such isolation in IaaS cloud services. We consider that a breach occurs when a communication channel is established between co-resident VMs and information is exchanged. A more precise definition and the underlying threat models are presented in section 1. The questions we are interested in answering are (i) can we guarantee VM co-location and, (ii) how can we use the shared hardware to bypass software isolation. To solve (i), research is needed to understand and take advantage of the mechanisms used by cloud service providers to allocate VMs to physical machines. Solving (ii) is not trivial either since physical machine specifications are usually hidden from the end-user. In such a context, it may be the case that co-resident VMs share fewer resources than expected, thus thwarting naive attacks.

The rest of this survey will be structured in the following way: in section 1 we present the relevant threat models and provide definitions, in section 2 we present the papers that solve (i) and led the way for this field of research, in section 3 we chronologically investigate attempts at solving (ii), finally in section 4 we assess and compare the techniques developed.

[⋆] This report is submitted as part requirement for the module COMP0057 Research in Information Security at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

1 Threat Models and Definitions

There are two threat models that we wish to study when considering the breaking of isolation boundaries in IaaS cloud services. These models are closely related and result respectively in *cross-VM covert channel attacks* and *cross-VM side channel attacks*. In this section, we present these models separately and show their similarities. In both models, the attacker is given the power to create multiple accounts with the service provider and launch multiple instances under each account. The attacker's accounts are standard in all ways and are limited to the tools provided by the cloud service. We do not limit the attacker's budget, however expenditures should be measured to assess practicality.

Cross-VM covert channel attacks – Under this first model, the attacker's goal is to obtain co-located VMs and exchange information between at least two VMs with a steady bit-rate. In doing so, the attacker establishes what Lampson defined as a *covert channel*: a channel "not intended for information transfer at all, such as the service program's effect on the system load." [7]. Should the attacker obtain co-located VMs, we will refer to one of them as the *sender* VM and the other as the *receiver* VM. Since these channels are often symmetrical, these labels are interchangeable.

Cross-VM side channel attacks – The second threat model is more restrictive. Here, the attacker is faced with a target VM that is not under her control. The attacker's goal is to co-locate one of her VMs with the target VM and extract meaningful information from the target. This scenario is akin to a covert channel where the target VM is a sender that does not cooperate with the receiver. In this case, the "receiver" (which we often call *attacker*) needs to spy on the "sender" (*victim*). This asymmetric channel is called a *side channel*. Notice that an attacker able to establish a side channel is also able to establish a covert channel. The converse proposition is not necessarily true.

2 Placement Vulnerabilities

2.1 The Breakthrough

In 2009, Ristenpart *et al.* published a breakthrough paper [13] that is unanimously cited as the birth of the field we are surveying here. In this study, the authors were able to obtain co-located VMs on Amazon EC2 with a non-negligible probability at a relatively low cost. Furthermore, they succeeded in both obtaining co-location between two random VMs and co-location with a target VM. The ability to achieve co-location with such probabilities is later defined in [16] as a *placement vulnerability*. It is seen as an exploitable feature of the service provider's *placement policy*: the policy by which the provider assigns a new VM to a physical machine.

To produce these results, Ristenpart *et al.* conducted many valuable experiments. First, they partially reverse-engineered the EC2 placement policy (although this has changed since). In doing so, they were able to map how certain user-chosen parameters affect an instance's IP address. Using this mapping, the authors developed heuristics to gain higher chances of achieving co-residence. Furthermore, the knowledge acquired about EC2's network topology allowed the authors to develop computationally cheap co-location tests. Finally, the authors established multiple covert-channels as a proof-of-concept that co-location is a security threat. These channels were extremely slow (under 1 bit-per-second) but acted as an invitation for other researchers to join their efforts.

2.2 Follow-up studies

In 2015, two simultaneous studies were published [16,20] that re-examined the results shown in [13]. Both papers point to the fact that Amazon had changed its placement policy and increased its security in response to [13]. Indeed, most of the network information leveraged by Ristenpart *et al.* had now been obfuscated from end-users [20], thus invalidating the heuristics and co-residency tests developed. However, both studies were able to once again achieve co-residence at low costs with high probability.

In [16], Varadarajan *et al.* formalised the security question that is VM co-location in the cloud. The authors provided formal mathematical definitions for the probability of obtaining co-located instances under a random placement policy and the cost of obtaining co-located instances using a specific attack strategy. They then identified the variables that are under the attacker's control and those that form part of the environment to perform experiments on the effects of each of the variables. Using those results, the authors offered multiple placement strategies and evaluated them on all three major IaaS cloud platforms. Unfortunately, the authors only presented their results relative to a benchmark; their strategies being at most ten times more likely to obtain co-residence than in a random placement policy setting, and up to $114 cheaper.

Xu and Wu [20] followed a different approach. Similarly to the original work of [13], they focused on network probing to guess an instance's location within the cloud. As a result, their study provides a more in-depth exploration of the changes in Amazon's network management. Their strategy to co-locate VMs can be seen as an extension of the heuristics developed in [13], with adaptations made to counter-act EC2's improved security. Results are presented in absolute value. When launching the cheapest VMs offered by Amazon, the authors achieved random co-residence in approximately 100 minutes while spending less than $5 [20]. Achieving targeted co-residence required to launch more instances. In this case, financial costs were higher, however the time necessary for the attack did not increase since fewer co-residency checks were needed (in this setting, the attacker only checks the instances launched, rather than checking for every instance pair as is the case in the random co-residence experiment).

Interestingly, both papers implemented the same testing method to check for co-residence. This method is based on the *memory bus contention* covert channel technique developed in [17] which we will cover in subsection 3.3. Both studies also explored a novel feature in EC2 known as Virtual Private Cloud (VPC). Using VPC, users may choose to logically isolate their network of instances from the rest of EC2. However, as both studies point out, this does not prevent instances from sharing hardware with other users. Achieving co-location with an instance residing in a VPC is still possible but may be up to three times more costly than a generic target [20].

As of today (2020), no further research has been published to update these results. IaaS cloud computing is a fast moving industry. There is very little doubt that service providers have already adapted their systems following these publications nearly five years ago. A follow-up study is probably overdue, ideally using the definitions developed in [16] to somewhat standardise results.

3 Communication Channels through Shared Hardware

In this section, we will investigate attacks that were attempted in the wild to answer objective (ii): how can we use the shared hardware to bypass software isolation. We present these attacks chronologically to give the reader a better sense of the challenges at stake and how the state-of-the-art improved.

Early years of the field were marked by clear trends in research efforts. Indeed the first few years were marked by attempts at building better covert channels with existing techniques [13,19,22] (subsection 3.2). Faced with practical challenges, the second trend was to search for different approaches [17,18] (subsection 3.3). In a third movement, researchers started developing new techniques and improving existing ones to overcome the initial challenges [2,3,9,21] (subsection 3.4). As the field grew and attracted new researchers and ideas, efforts were less clustered into trends and all three approaches were adopted concurrently [1,10,12] (subsection 3.5).

Before we investigate each of the periods, we introduce some necessary technical background in subsection 3.1.

3.1 Technical Background

Establishing Channels – On a very high level, the method used to create a communication channel through shared hardware is generally the same: find a process that can be initiated by one VM, that will cause a measurable change in hardware for the second VM to observe. In the case of covert channels, this can be any process; in the case of side channels, the process must be part of the victim's naturally occurring execution.

There are two properties of interest when building cross-VM communication channels. The first one is the communication speed, which can be measured as its average bit-rate. In theory, attacks targeting faster hardware should allow faster channels. The second property is error-resistance, which is usually measured by the error rate. As we will see in the following section, there are many possible sources of errors. Understanding and characterising these will dramatically improve performance.

Processors, Cores and Caches: a Quick Presentation of our Protagonists – As mentioned above, attacking faster hardware components brings the hope of better results. Therefore, many attacks focused on *central processing units* (*CPU*, or colloquially *processor*). Modern CPUs have multiple *cores*, which are smaller processing units that allow the CPU to process tasks in parallel.

Processors are much faster than memory and therefore are often stuck waiting for memory fetches to finish executing. To minimise this issue, CPUs are equipped with built-in memory units called *caches*. These units are much faster than regular memory, however they are much more expensive to produce. As a result, they often have small capacities. In a cache, data is store in *lines* of l bytes. Line are themselves arranged into *sets*.

To achieve an optimal balance between performance and cost, caches are often arranged into multiple layers with different speed/cost trade-offs. Recent Intel architectures make use of three levels of cache L1, L2 and L3. These are ordered from fastest (L1) to slowest (L3). In virtue of being the last level of cache, L3 is adequately named the Last Level Cache (LLC). When the processor tries to access a value in memory, it starts by checking the first and fastest layer of cache (L1). If the value is present in the cache (a *cache hit*) the memory access is completed, if the value is absent from the cache (a *cache miss*) the processor checks the next level of cache until it eventually reaches the main memory [10].

In Intel multi-core processors, each core has its own private L1 and L2 caches. The LLC is connected to all cores and is fully accessible by all of them [10]. Furthermore, in some architectures, the LLC is said to be *inclusive*: in that case, the LLC is a superset of the earlier layers of cache.

Throughout this survey, unless mentioned otherwise, a processor will refer to an Intel x86 processor. The exact model under study often depends on the year in which a specific experiment was run. We discuss this heavy bias towards Intel processors in section 4.

3.2 2009-12: The *Prime+Probe* era

As mentioned in section 2, the first attack to be performed in the cloud was the proof-of-concept covert channels presented in [13]. Ristenpart *et al.* [13] targeted the L2 cache as a shared resource and applied a method developed in [15] called *Prime+Probe*. Here we assume that two parties, the sender and the receiver, have access to a shared cache. The receiver first *primes* a target *cache set*: the *set* is filled up by values known to the receiver. The sender then selectively evicts certain *lines* of the *set* by loading their own data. The receiver can then try to re-access their data, thus *probing* the *set* to check if the values from the *prime* step are still present. By accessing these values and measuring the time taken for a memory fetch, the receiver will observe fast fetches for data that is still present (cache hits) and slow fetches for data that had been evicted (cache misses). Both parties can therefore agree to a protocol where an eviction represents a 1 and the lack thereof represents a 0. Notice that the *Probe* step fills up the *cache set* with data known to the receiver and therefore doubles up as a *Prime* step to transmit the next symbol.

The covert channel implemented in [13] uses a shared L2 cache and achieves a bit rate of 0.2 bits-per-second (bps). The authors however clarify that no attempt at optimisation were made.

In 2011, Xu *et al.* [19] implement the same L2 cache covert channel using *Prime+Probe* and proceed to optimise the protocol. Using their refined protocol, they approximate the maximum bit-rate that such a channel can achieve in a laboratory environment. While their calculations show that bit-rates slightly over 200bps are achievable, their experiments on EC2 yield results order of magnitude below. Indeed their fastest channel only yielded 10.46 bps with an error rate of 28.13%. The authors were able to obtain an error-free channel (0% error rate) by sending single bits multiple times. This however reduced the effective bit-rate even further to 1.27 bps.

Xu *et al.* [19] identify many of the challenges that slowed down their attack. Firstly, relying on the L2 cache implies that both parties need to be executing on the same core. While [13] showed that VM co-location could be achieved, this does not guarantee that both VMs will be scheduled to use the same core for an extended period of time. The attack therefore suffers from these changes in scheduling. They conclude that the attack could be useful in the form of a side-channel to extract short secrets such as cryptographic keys. This approach became the dominant one for the next few years.

The first study to attempt such a cross-VM side channel attack was [22]. Implementing a variant of the *Prime+Probe* method, the authors managed to spy on a decryption process and exfiltrate the private key. However, their mitigations against the issues faced by Xu *et al.* [19] were a collection of ad-hoc processes. While the attack worked in a very specific setting, it could definitely not be transferred into the cloud, nor would it be practical on a machine outside of the laboratory.

3.3 2012-13: Exploring hardware resources

In 2012, as a caveat to the doctrine described above (build side channels to steal short but extremely important information), Wu *et al.* [17] attempted to render covert channel attacks applicable in more restrictive settings, with better bandwidth and less errors. First, they identified with greater precision the sources of errors that hindered the success of [19]. They point to three challenges: *addressing uncertainty*, *scheduling uncertainty* and *cache limitations*. We briefly summarise these issues and then present the authors' solution.

Addressing uncertainty refers to the many layers of virtualisation between physical memory addresses and the addresses that are actually shown to the end-user. Indeed, the user's operating system (OS) typically implements memory virtualisation. However, the addresses seen by the user's OS are themselves virtual addresses provided by the hypervisor. There are therefore at least two layers of virtualisation between a user-accessible address and a physical address. This is true for both the sender and receiver, thus adding complexity.

Scheduling uncertainty is a harder problem to solve. Indeed, the scheduling algorithm used by the cloud provider is mostly unknown to the attacker. However, attacks such as those presented in [13], [19] and [22] require that both parties are executed in a strict round-robin. Out-of-order execution or the execution of a third party could completely thwart these attacks.

Finally, *cache limitations* addresses the fact that a shared cache is not always available. Wu *et al.* [17] point out the fact that L2 caches are not shared between cores. Furthermore, cloud providers make use of multi-processor machines, thus removing the hope for a shared LLC.

Wu *et al.* [17] solve these issues by considering a different hardware resource, the memory bus, and executing the sender and receiver in parallel. In older multi-processor systems, a memory bus would connect all processors to all available memory units. In modern systems, the memory bus is not implemented by a single connection, however all components remain interconnected. The authors noticed that executing atomic memory instructions cause the memory bus (or the equivalent inter-connect) to enter a state of contention, where only the issuer of the instruction can access it. This contention is observable system-wide since all other users are temporarily prevented from accessing memory. The two states of the memory bus (contended and contention-free) can therefore be used to encode binary information.

The authors developed a protocol that takes advantage of this property and implemented it on Amazon EC2. Their *in vivo* experiment yielded a bit-rate of 107.9 bps with an error rate of 0.75%. This is a drastic improvement on previous state-of-the-art. Furthermore, this method provides greater flexibility with regards to the shared hardware. Indeed the channel stands as long as the sender and receiver are co-resident and executed at the same time, regardless of which processor and which core they are operating on. This property explains why the *memory bus contention* method was used in both [16] and [20] as a robust co-residency test.

Xiao *et al.* [18] make a similar attempt at deviating from cache-based attacks to solve the three challenges described above. Here, the authors make use of shared memory contents to take advantage of a process known as *memory de-duplication.*

Memory de-duplication is a resource optimisation process whereby the hypervisor scans the memory contents of its guest VMs for identical content. Should two or more VMs share an entire page of memory, the hypervisor only saves one version of it and grants access to all the relevant VMs. If one VM modifies the page's contents, the hypervisor creates a private copy for that VM and implements the requested changes; this is known as a *copy-on-write* policy. This however introduces a timing vulnerability as first shown in [14]. Indeed, *copy-on-write* implies that a write operation will take longer on a memory page that is shared with other VMs. The authors of [14] use this property to detect the presence of other VMs.

Xiao *et al.* [18] use a technique similar to *Prime+Probe* but apply it to memory pages. Both the sender and receiver VM load a large file into memory, occupying multiple pages. After a short waiting time, the hypervisor will have noticed the identical memory pages and de-duplicated them. The sender VM selectively edits parts of the file that belong to specific pages. The receiver VM then tries to write to the entire file page-by-page and times these operations. Modified pages will take longer to write to. Once again this timing difference allows the receiver VM to deduce the sender's activity, thus receiving binary information (modified vs not modified pages).

While this process circumvents the issues of *addressing uncertainty*, *scheduling uncertainty* and *cache limitations*, it yields a much slower attack. Indeed with every round, both parties need to load a large file and wait for the hypervisor to apply memory de-duplication. As a result, the authors measured a maximum bit-rate of 90 bps in a laboratory environment. Thus the *memory de-duplication* attack is an improvement on previous cache-based attacks, however this solution is less applicable and slower than that presented in [17].

3.4 2014-16: New and improved cache-based attacks

Returning to the doctrine of building small capacity channels that steal important secrets, research once again focused on the processor cache. Researchers were determined to build attacks that could do the most damage rather than improving the communication process.

In light of this new focus, Yarom and Falkner [21] present a novel technique to build cross-VM side channels. The so-called *Flush+Reload* technique allows an attacker to spy on a process executed by the victim on the condition that both VMs have access to a shared page of memory. This technique relies on flushing *lines* from the LLC. Recall that the LLC is the last level cache and is shared across all cores. Furthermore, in Intel processors the LLC is inclusive of earlier layers of cache: any data that is present in a core's private L1 and L2 must also be present in the LLC. To maintain inclusiveness when a *line* is flushed from the LLC, that same *line* is also flushed from earlier levels of cache. Therefore *Flush+Reload* is a cache attack that partly mitigates the *cache limitations* described in [17] by working across cores. Furthermore, it operates at a finer level of granularity than *Prime+Probe*, using *cache lines* rather than *cache sets*.

The attack works as follows. Assume an attacker VM and a victim VM are co-resident and share a page of memory – in the case presented in [21], they share the instructions to run a square-and-multiply algorithm as part of a standard cryptographic library. The attacker starts by flushing *cache lines* that correspond to the execution of the shared code from the LLC. After a short wait, the attacker tries to reload those same instructions from the shared memory page. If the attacker observes a cache hit, she can deduce that the victim ran those instructions and therefore storing them in the LLC. On the other hand if the attacker observes a cache miss, she can deduce that the code was not executed. This is particularly hurtful for implementations of RSA where the execution of instructions (square, multiply and/or reduce) is dependent on individual bits of a private key.

While this attack does work across cores in virtualised environments, it faces a couple of limitations that make it impractical in the cloud. Firstly, it is assumed that both VMs share a page of memory. Although this is likely in a standard setting where memory de-duplication is enabled, this feature had been removed from IaaS hypervisors as a security measure following the publications of [14] and [18]. Secondly, the attack is sensitive to interference on the LLC. Should a third party make use of the LLC at the same time as the victim, the attacker may register false readings.

The invention of *Flush+Reload* is nonetheless significant. Indeed, this technique is faster to execute than the original *Prime+Probe* method [21] as it requires fewer memory accesses with each round. With this discovery comes the promise of further improving cache-based attacks. The technique is later implemented in an cross-VM side channel attack against an implementation of AES [5]. However, since [5] does not present an improvement of the technique it is outside of the scope of this review. Similarly, Zhang *et al.* [23] implement this technique on a platform-as-a-service cloud, but do not contribute to answering the subject of this survey.

In 2015, Liu *et al.* [9] and Irazoqui *et al.* [3] simultaneously adapted the *Prime+Probe* technique to work across cores via the LLC and without the need for shared memory contents. This improvement is especially important as it sheds off another requirement from cache-based attacks.

The main challenge that prevented *Prime+Probe* from being applied in the LLC is its size. Indeed, probing the entirety of the LLC would take too long to monitor any meaningful operation on the victim's side [9]. On the other hand, *Flush+Reload* thrives in the LLC since the shared memory contents allow the attacker to only monitor specific *lines* of the LLC.

In both works [3,9], the authors reverse-engineered the undisclosed mapping from memory to the LLC to be able to monitor specific areas of the LLC. Using this mapping, a sender and receiver can target a subset of the LLC to apply the *Prime+Probe* technique we have already described. In this case, the LLC's size is no longer an issue as long as both parties can agree on a region to target. Thus we have the first cache-based attack that works across cores and does not require shared memory.

The *Flush+Reload* technique was further improved in 2016 by Gruss *et al.* [2] who introduced the *Flush+Flush* technique. This attack works in the same way as *Flush+Reload*, however here the attacker VM simply loops flushing operations and times their execution. Indeed, the authors noticed that a flush on an already-flushed line of cache is executed faster than a flush on a populated line of cache. Thus *Flush+Flush* is similar in all ways to *Flush+Reload*, with the exception that it is faster to execute. However, the difference in timing of these flush operations may be harder to measure than cache hits/misses and lead to higher error rates.

To illustrate their contribution, Gruss *et al.* [2] built covert channels through the LLC using all three cache attacks we have seen so far (improved *Prime+Probe*, *Flush+Reload* and *Flush+Flush*). Results show that *Flush*-based techniques are faster than the LLC *Prime+Probe* [2]. Under certain conditions, *Flush+Flush* is shown to establish the fastest covert channel – up to 496 Kbps – observed until then in a virtualised, local environment (as opposed to "in the cloud").

Notice however, that *Flush+Flush* suffers from the same requirements as *Flush+Reload*, namely that the attacker and victim (or receiver and sender) share memory contents. Indeed, although the attacker does not access the memory contents in this technique, she still needs to know which line(s) of cache to flush. This information can only be gathered by guessing (unlikely), or by having a copy of the memory contents to load herself. Thus, while *Flush+Reload* and *Flush+Flush* are faster than the newly improved *Prime+Probe*, they are both impractical on IaaS clouds in their current states.

3.5 2015-17: Try building, try something different, keep improving

In this final, more recent period of research, all three approaches explored so far are attempted concurrently.

In 2015, Maurice *et al.* [10] returned to the "try building" approach and attempted to establish a cross-core cache-based covert channel. By now, we hope that the reader recognises that a cache-based cross-core attack will likely make use of the LLC. Indeed this is the case in [10]. Note that this paper was published before the improved *Prime+Probe* method of [3] and [9]. Maurice *et al.* [10] therefore make their own attempt at adapting a *Prime+Probe*-style technique that does not require shared memory to the LLC. However, their approach is less successful than those of [3] and [9], and is therefore of lesser interest in answering our survey's questions.

The publication of [12] by Pessl *et al.* in 2016 marks the first attempt since 2012-13 at deviating from cache-based attacks – "try something different". Indeed this attack is most comparable to the *memory bus* covert channel presented in [17]. Here the authors target DRAM (dynamic random access memory) as a shared resource in multi-processor systems. In order to better present this attack, we must make an aside to describe typical DRAM hierarchy.

The following explanation is adapted from the background section of [12]. In multi-processor settings, each processor is equipped with a memory controller and has a direct channel to a dual in-line memory module (DIMM). Naturally, the processors are interconnected otherwise this would not be a multiprocessor system. Each DIMM is then divided into ranks, then banks and finally rows. A memory address specifies a physical location: this means specifying which row, of which bank, in which rank, of what DIMM. In Intel systems, the mapping from a memory address to a physical location is performed using an undisclosed function. This mapping is known as *DRAM addressing*.

Pessl *et al.* [12] develop a method to remotely reverse-engineer a system's *DRAM addressing* function. Having recovered this mapping, an attacker can control the specific location in silicon where her data is saved. A sender and receiver running natively on the same machine can use this mapping to ensure that they are saving data to rows belonging to the same bank (this implies within the same rank of the same DIMM).

The second observation made by the authors is that each bank has a *row buffer* [12]. A row requested by a processor is first loaded into the *row buffer* and then sent across the memory bus. Although it is not its purpose, this *row buffer* behaves like a cache: sending the same row a second time is faster, whereas sending a different row creates a *row conflict* and yields a slower transmission. Pessl *et al.* [12] therefore develop a protocol that takes advantage of the two states of the *row buffer*.

While not acknowledged in the paper, their protocol is in fact an application of the *Prime+Probe* method to the row buffer. First, the receiver *primes* the *row buffer* by requesting its data from the DRAM unit, thus filling up the *row buffer*. The sender can then send a 0 by doing nothing, or send a 1 by accessing the data it has stored in the same bank. Doing so will create a *row conflict* which the receiver will observe in the *probe* step. As we have already mentioned this *probe* step acts as a *prime* step for the next symbols.

Performing the attack in a virtualised environment introduces the well-known issue of *addressing uncertainty*. In this case, the method for reverse-engineering *DRAM addressing* only works under specific hypervisor settings. Running two VMs on their own servers with the necessary hypervisor settings, Pessl *et al.* [12] were able to use their technique to build a covert channel yielding a raw bit-rate of 596kbps (kilobits-per-second) for a 0.4% bit-error rate.

These results are orders of magnitude higher than the only other cross-processor attack presented in [17]. The first reason for this is that the DRAM addressing attack can be parallelised. Indeed while [17] makes use of the unique memory bus, Pessl *et al.* [12] use *row buffers* of which there are multiple instances within a system. Recall that each bank has a row buffer, and that each rank has multiple banks, and each DIMM has multiple ranks. Thus for d DIMMs, r ranks and b banks, there are $d \times r \times b$ row buffers that can be used in parallel. Secondly, we can assume that part of the speed-up observed is in fact due to more recent hardware.

Overall, this attack is the second cross-processor, cross-core attack that does not require shared memory. However it requires a specific configuration of the hypervisor in order to be attempted across VMs. Such a configuration is out of the attacker's control in the threat model we study in this survey.

Recent papers have also attempted to improve the existing techniques. In [1], Disselkoen *et al.* present an improved version of *Prime+Probe*, which they name *Prime+Abort*. This method removes the need for precise timing measurements to monitor cache hits and misses. However, *Prime+Abort* has not yet been implemented in a cross-VM channel, and therefore remains out of the scope of our review for the time being.

The most promising attack to be attempted on an IaaS cloud is the one presented in [11]. Here, Maurice *et al.* [11] make use of the state-of-the-art *Prime+Probe* technique presented in [3] and

[9]. In this paper, the authors focused on using existing techniques to build a robust cross-VM channel in the cloud. Assuming that obtaining co-location is possible with the required effort [13,16,20], they instead obtained co-located VMs by requiring a dedicated machine from Amazon EC2. This functionality is offered by the cloud platform in exchange of a cost premium. Note that this attack relies on *Prime+Probe*, therefore requires that both the sender and receiver are executed on the same CPU.

In addition to implementing the *Prime+Probe* attack, Maurice *et al.* [11] provide a protocol for the sender and receiver VMs to agree on cache sets to communicate through, incidentally solving *addressing uncertainty*. The channel is then implemented on Amazon EC2 using various non-trivial error correction methods. Furthermore, Maurice *et al.* [11] implement a synchronisation mechanism to solve *scheduling uncertainty* and the errors that it generates. Under various stress conditions, their channel maintains a steady bit-rate between 37 and 45 kilobytes-per-second with a 0.00% error rate. To showcase their achievements, the authors implement a fully functioning SSH connection across the two VMs.

4 Comparison and Discussion

In this final section, we wish to summarise the research achievements in the field, provide comparisons between various techniques and contemplate the possibility of future work.

Attack summary – To obtain a functional cross-VM communication channel on an IaaS cloud, an attacker must first force co-location. This was shown to be possible in [13,16,20]. Then, the attacker must find a process that meets the following requirements:

a) Can be initiated from within an unprivileged VM
b) Works across cores
c) Does not require shared memory contents between both parties
d) Works across processors

Table 1 aggregates all the techniques presented above and displays how they respond to these requirements

Year Developed	Method	Used in	a)	b)	c)	d)
2009	*Prime+Probe* v1	[13,19,22]	✓		✓	
2012	Memory bus contention	[17]	✓	✓	✓	✓
2013	Memory de-duplication	[18]	✓	✓		✓
2014	*Flush+Reload*	[5,21,23]	✓	✓		
2015	*Prime+Probe* v2	[3,9,10,11]	✓	✓	✓	
2016	*Flush+Flush*	[2]	✓	✓		
2016	DRAM addressing	[12]		✓	✓	✓
2017	*Prime+Abort*	TBD	TBD	TBD	TBD	TBD

Table 1. Summary table of attempted methods

Notice that out of all the attacks attempted, only one meets all four requirements ([17]). However, a number of the other studies have been able to implement either a covert channel or a side channel on an IaaS cloud ([11,13,19]). This is because requirements b) and d) can be worked around with favourable execution conditions. An attacker may, per-chance, find herself sharing the same processor or the same core as her victim. Depending on the scenario, the attacker could even flood a machine with instances such as to guarantee that at least one VM shares the same core and/or processor as the victim. This is in fact the method applied in [11] to circumvent the limitations of the *Prime+Probe* method.

Overall, researchers have been successful in establishing cross-VM covert channels. Doing so in the cloud introduces a variety of challenges, however the results shown in [17] and later in [11] prove the existence of an exploitable (and exploited!) vulnerability.

The problem of quantitative comparisons – Providing a quantitative comparison of these channels, or individual techniques, is less obvious than it would seem. Indeed, the reader may have already noticed that bit-rates seem to vary widely from one study to the next. All of the attacks presented make use of hardware resources to establish a channel. Therefore, the variation in hardware from one study to the next creates large variations in bit-rates. As such, we recommend to only focus on bit-rates for channels that are implemented against a standard testbed: any commercial, public IaaS cloud. Bit-rates from laboratory experiments are of interest only if a comparable technique was implemented on the same hardware using the same communication protocol.

The hard(ware) question – All of these studies have heavily focused on Intel processors. This is partly due to their widespread in consumer and commercial hardware. However, another explanation is that other processors were harder to attack. As pointed out in [4], commercial-grade AMD processor have more cores-per-processor than Intel's CPUs. Furthermore, in AMD architectures the LLC is rarely shared between cores and if it is, is not inclusive of earlier layers of cache. These properties imply that cache-based attacks such as the ones research had focused on would not function on AMD processors. Similar issues arise in ARM processors.

Other IaaS providers – Research in this field has also been heavily biased towards Amazon EC2. Indeed, at the time of publication of Ristenpart *et al.*'s [13] seminal paper, EC2 was the only available public cloud. Subsequent papers focused on it since [13] provided a proof-of-concept for that service only. The follow-up studies to [13], namely [16] and [20], both exposed placement vulnerabilities in Microsoft Azure and Google Compute Engine. However, to our knowledge, no attempt has been made at to measure covert or side channels in those IaaS clouds.

A final note – Since 2017, very few relevant papers have been published on this topic. However, this must not be mistaken for a lack of engagement by the research community. In 2018, the now-famous papers describing the *Spectre* [6] and *Meltdown* [8] attacks were published. Authors of these papers include many researchers whose work is featured here: Daniel Gruss [2,11,12], Stefan Mangard [2,11,12], Michael Schwartz [11,12] and Yuval Yarom [9,21]. Indeed the expertise gathered by building these covert and side channel attacks partly enabled the discovery of these two security flaws.

Furthermore, research papers have recently focused on detecting such channels. While the processes at stake are very similar to the offensive techniques we explored, these publications are out of the scope of our survey.

Conclusion

Over the past decade, researchers have heavily scrutinised the IaaS cloud infrastructure for security issues. In 2009, Ristenpart *et al.* [13] opened a field of research by showing that Amazon's placement policy was vulnerable to attacks, enabling a malicious party to obtain co-location. Following their proof-of-concept, offensive techniques were developed at a steady pace to break down isolation boundaries between virtual machines on IaaS clouds. Each technique provided new insight into the depths of processor architecture and how to exploit it. Very few attacks have managed to present a fully universal attack vector. Instead many have achieved the desired goal under special circumstances or with slow communication ability. These attack undeniably show that cloud security has much to improve.

Naturally, all these papers point towards mitigations against such attacks. While modifying hardware specifications is costly for cloud providers, restricting the access to certain resources may be a good solution. Stricter hypervisor settings and a stronger placement policy will also contribute to reinforcing cloud security. Finally, clients wary of such attacks already have the option to physically isolate their VMs from the rest of the cloud service by paying a premium to obtain a dedicated machine.

Suggestions for future work in the field include: a renewed study of placement policies, an assessment of the covert channel threat across all three major IaaS cloud providers, a study of these attacks against processors from a wider range of manufacturers and, improvements of promising techniques such as the *DRAM addressing* attack or the *Prime+Abort* method.

References

1. Disselkoen, C., Kohlbrenner, D., Porter, L., Tullsen, D.: Prime+abort: A timer-free high-precision l3 cache attack using intel TSX. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 51–67. USENIX Association (2017), https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/disselkoen
2. Gruss, D., Maurice, C., Wagner, K., Mangard, S.: Flush+flush: A fast and stealthy cache attack. In: Caballero, J., Zurutuza, U., Rodríguez, R. (eds.) Detection of Intrusions and Malware, and Vulnerability Assessment, vol. 9721. Springer (2016)
3. Irazoqui, G., Eisenbarth, T., Sunar, B.: S$A: A shared cache attack that works across cores and defies vm sandboxing – and its application to aes. In: 2015 IEEE Symposium on Security and Privacy. pp. 591–604 (2015)
4. Irazoqui, G., Eisenbarth, T., Sunar, B.: Cross processor cache attacks. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. pp. 353–364 (2016), https://doi.org/10.1145/2897845.2897867
5. Irazoqui, G., Inci, M.S., Eisenbarth, T., Sunar, B.: Wait a minute! a fast, cross-vm attack on aes. Cryptology ePrint Archive, Report 2014/435 (2014), https://eprint.iacr.org/2014/435
6. Kocher, P., Horn, J., Fogh, A., , Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y.: Spectre attacks: Exploiting speculative execution. In: 40th IEEE Symposium on Security and Privacy (S&P'19) (2019)
7. Lampson, B.: A note on the confinement problem. Association for Computing Machinery, Inc. (1973), https://www.microsoft.com/en-us/research/publication/a-note-on-the-confinement-problem/
8. Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M.: Meltdown: Reading kernel memory from user space. In: 27th USENIX Security Symposium (USENIX Security 18) (2018)
9. Liu, F., Yarom, Y., Ge, Q., Heiser, G., Lee, R.B.: Last-level cache side-channel attacks are practical. In: 2015 IEEE Symposium on Security and Privacy. pp. 605–622 (2015)
10. Maurice, C., Neumann, C., Heen, O., Francillon, A.: C5: Cross-cores cache covert channel. In: Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9148. pp. 46–64 (2015), https://doi.org/10.1007/978-3-319-20550-2_3
11. Maurice, C., Weber, M., Schwarz, M., Giner, L., Gruss, D., Boano, C.A., Mangard, S., Römer, K.: Hello from the other side: SSH over robust cache covert channels in the cloud. NDSS Symposium 2017 (2017), https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/hello-other-side-ssh-over-robust-cache-covert-channels-cloud/
12. Pessl, P., Gruss, D., Maurice, C., Schwarz, M., Mangard, S.: DRAMA: Exploiting DRAM addressing for cross-cpu attacks. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 565–581 (2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl
13. Ristenpart, T., Tromer, E., Shacham, H., 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. pp. 199–212 (2009), https://doi.org/10.1145/1653662.1653687
14. Suzaki, K., Iijima, K., Yagi, T., Artho, C.: Memory deduplication as a threat to the guest os. In: Proceedings of the Fourth European Workshop on System Security. Association for Computing Machinery (2011), https://doi.org/10.1145/1972551.1972552
15. Tromer, E., Osvik, D., Shamir, A.: Efficient cache attacks on aes, and countermeasures. In: Journal of Cryptology 23. pp. 37–71 (2009)
16. Varadarajan, V., Zhang, Y., Ristenpart, T., Swift, M.: A placement vulnerability study in multi-tenant public clouds. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 913–928 (2015), https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/varadarajan
17. Wu, Z., Xu, Z., Wang, H.: Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In: Proceedings of the 21st USENIX Security Symposium (USENIX Security 12). pp. 159–173 (2012), https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/wu
18. Xiao, J., Xu, Z., Huang, H., Wang, H.: Security implications of memory deduplication in a virtualized environment. In: Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'13). pp. 1–12 (2013)
19. Xu, Y., Bailey, M., Jahanian, F., Joshi, K., Hiltunen, M., Schlichting, R.: An exploration of l2 cache covert channels in virtualized environments. In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop. pp. 29–40 (2011), https://doi.org/10.1145/2046660.2046670
20. Xu, Z., Wang, H., Wu, Z.: A measurement study on co-residence threat inside the cloud. In: Proceedings of the 24th USENIX Conference on Security Symposium. pp. 929–944 (2015)
21. Yarom, Y., Falkner, K.: Flush+reload: A high resolution, low noise, l3 cache side-channel attack. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 719–732 (2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

22. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-vm side channels and their use to extract private keys. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 305–316 (2012), https://doi.org/10.1145/2382196.2382230
23. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-tenant side-channel attacks in paas clouds. In: CCS '14 (2014)