THEME ARTICLE: Top Picks

Uncovering the Security Implications of Cloud Multi-Tenancy with Bolt

Christina Delimitrou
Cornell University

Christos Kozyrakis
Stanford University

Cloud providers routinely schedule multiple applications per physical host to increase efficiency. The resulting interference on shared resources often leads to performance degradation and, more importantly, security vulnerabilities. Interference can leak important information about an application, ranging from a service's placement to confidential data, such as private keys. We present Bolt, a practical system that accurately detects the type and characteristics of applications sharing a cloud platform based on the interference an adversary sees on shared resources. Bolt leverages online data mining techniques that only require 2-5 seconds for detection. In a multi-user study on Amazon Elastic Compute Cloud (EC2), Bolt correctly identifies the characteristics of 385 out of a set of 436 diverse workloads. Extracting this information enables a wide spectrum of previously impractical cloud attacks, including denial of service (DoS) attacks that increase tail latency by 140X, as well as resource freeing attacks (RFAs), and co-residency attacks. Finally, we show that, while advanced isolation mechanisms such as cache partitioning lower detection accuracy, they are insufficient to eliminate these vulnerabilities altogether. To do so, one must either disallow core sharing or allow it only between threads of the same application, leading to significant inefficiencies and performance penalties.

The end of Dennard scaling, and the subsequent end of exponentially increasing performance at constant power consumption it provided, has promoted efficiency to a first-class requirement for both large- and small-scale systems.[1] Efficient use of hardware resources translates to systems that scale better as user demand increases. The need for efficiency becomes even more obvious in large-scale datacenters that consist of tens or hundreds of thousands of servers.[2] To improve

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

resource efficiency, both private and public clouds employ multi-tenancy, which results in system resources being shared across multiple applications.[1,2,3]

Unfortunately, multi-tenancy leads to resource interference, which has performance[2,3] and security implications; this has prompted an extensive amount of work on side-channel, DoS, and data exploitation attacks that leverage the lack of strict isolation to extract confidential information from victim applications, such as passwords and keys.[4,5] For a review of this related work, see the "Research Related to Bolt" sidebar.

In our article for the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17),[6] we showed that cloud multi-tenancy also enables a new class of security vulnerabilities wherein an adversary can leverage practical data mining techniques to quickly and accurately extract information about the type and characteristics of victim applications. Once attackers obtain this information, they can launch severe, inexpensive, and hard-to-detect performance attacks against applications with which they are sharing resources.

The key insight of Bolt is that—instead of relying on detailed timing information or hardware instrumentation (which, in the past, have enabled side-channel attacks)—it employs a data-driven approach. Specifically, it uses a set of practical, online data mining techniques to determine the pressure a victim application introduces in one of several shared system resources. Having this information, the runtime can identify the type and functionality of a victim application, as well as which resources are critical for its performance. This knowledge enables many performance attacks, including host-based DoS attacks, RFAs, and VM pinpointing attacks. The key insight in these attacks is that the runtime uses the resources a victim is sensitive to against it, causing it to withstand significantly degraded performance without necessarily saturating system resources. This means that the attacks Bolt enables are resilient to the auto-scaling techniques many cloud providers employ, such as CPU utilization-triggered VM migration. We validated Bolt in a controlled environment and in a multi-user study across Stanford University and Cornell University, using a shared Amazon EC2 cluster with 200 servers. In all cases, the runtime was able to correctly identify the majority of victims and negatively impact their performance. Finally, we evaluated to what extent current isolation techniques can mitigate these attacks, and we found that, although beneficial, they are insufficient to altogether prevent them.

The trade-off between performance, security, and resource utilization means that cloud providers and users currently have two options. The first is to forgo resource efficiency to ensure good performance and secure applications against malicious users, which in turn results in poor cloud scalability, increased cost, and reduced cloud adoption. The alternative is to sacrifice performance and security in favor of high resource utilization, which again disincentivizes cloud usage. The inflection point at which computer systems are—with device scalability—slowing down, while application performance and security demands grow, means that neither of these two options is viable. Instead, modern cloud systems need strict and scalable resource isolation mechanisms in hardware and software that preserve application performance and security without sacrificing high resource utilization.

THREAT MODEL

Bolt targets infrastructure as a service (IaaS) providers that operate public clouds with any number of mutually untrusting users. Applications from multiple users can be co-scheduled on the same physical node to increase the system's resource efficiency; however, users have no control over where their applications are placed and no *a priori* information on the placement and characteristics of other users' applications. The threat model also assumes that the cloud provider is passive and impartial, meaning that they do not assist the adversary nor employ any additional techniques to deter their attacks, beyond isolation mechanisms that are readily available in containers or VMs.

There are two types of users under this threat model: adversarial and victim. A VM from an adversarial user has the goal of determining the nature and characteristics of any applications co-scheduled on the same physical host, as well as negatively impacting their performance. Adversarial VMs start with no knowledge of co-scheduled workloads. Victim users, on the other hand, are normal VMs scheduled on a physical host. Friendly VMs do not attempt to determine the existence and characteristics of other co-scheduled VMs. They also do not employ any schemes to prevent detection, such as memory pattern obfuscation.

ONLINE ADVERSARIAL DATA MINING

Figure 1 shows an overview of the application detection process. The key requirement for Bolt to be effective is sharing system resources with one or more victims. To achieve this, Bolt instantiates an adversarial VM on the same physical host as the victim(s). Bolt uses this VM to measure the pressure the victim(s) exert on different shared resources. The measurement process uses a set of contentious micro-benchmarks of tunable intensity, each targeting a different physical resource. Once Bolt obtains this sparse signal of the victim's interference profile, it leverages a hybrid recommender system to infer:

- First, the pressure the victim puts in all remaining, non-profiled resources and,
- Second, the type and characteristics of the victim application.

For the first step, Bolt uses a collaborative filtering approach that assumes no contextual information on the characteristics of the victim, simply finding similarities between resource profiles of profiled applications over time. The collaborative filtering recommender relies on matrix factorization using singular value decomposition (SVD), as well as latent factor models using stochastic gradient descent (SGD).

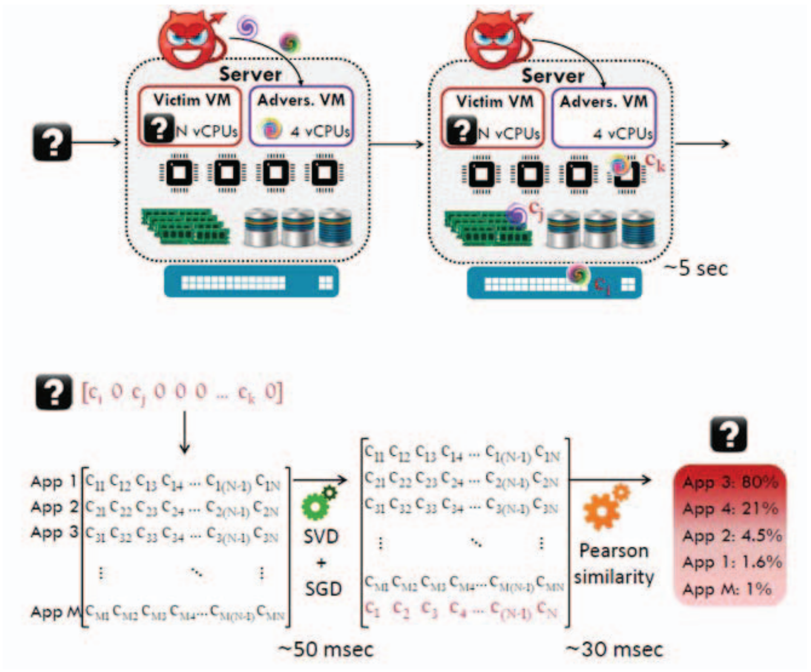


Figure 1. Overview of the application detection process. Bolt first measures the pressure co-residents place on shared resources, and then uses data mining to determine the type and characteristics of co-scheduled applications.

Once the victim's full resource profile is recovered, a content-based recommender system uses weighted Pearson correlation coefficients to compute the similarity between the resource profile of the new victim and the profiles of applications the system has previously seen. The content-based system uses these similarity scores to label the victim in terms of its application class, functionality, and, in many cases, user-load and dataset characteristics. The combination of collaborative filtering and content-based recommendation achieves the best of both worlds: the scalability, low overheads, and relaxed sparsity constraints of collaborative filtering, with the contextual information and high detection accuracy of content-based systems.

The output of the hybrid recommender is a vector of applications the victim is similar to, in decreasing order of similarity. For example, a victim might be 65 percent similar to a memcached workload, 18 percent similar to a Spark job running PageRank, 10 percent similar to a Hadoop job running a support vector machine (SVM) classifier, and 3 percent similar to a Hadoop job running k-means. The 95th percentile of the recommender's end-to-end latency is 80 ms. Apart from labeling a victim, this analysis also provides information on the resources the victim is sensitive to, enabling several practical performance attacks, which we describe later.

Challenges in Detection

Detection is straightforward when there is only one victim per physical host, but it becomes more challenging when multi-tenancy is extensive. To disentangle the resource behavior of multiple co-scheduled victims, we have developed a shutter-based profiling mode, which involves frequent and very brief profiling phases (10-50 ms each). The goal of this mode is to capture the imprint of one application's resource pressure at a time, assuming fluctuations in user load, much like a high-speed camera shutter attempts to capture a fast-moving object. A second challenge for detection is the frequent changes in application behavior and load, or even the case where the same VM or container is used for different applications over time—a common use case for many cloud users. To address this challenge, Bolt periodically re-profiles the victims, and if it detects a substantial change in resource characteristics, it re-classifies them and potentially adjusts its performance attacks to their new resource profiles.

DETECTION VALIDATION

We have extensively validated Bolt's detection accuracy in a controlled setting of a 40-server cluster with 108 diverse applications, including batch analytics in Hadoop and Spark and latency-critical, interactive services, such as memcached, Cassandra, and webservers. For each application type, there are several workloads with respect to the algorithmic choice, framework version, dataset, and input load pattern. Friendly applications are scheduled using a least-loaded (LL) scheduler that allocates resources on the machines with the most available compute, memory, and storage. All workloads are provisioned for peak requirements to reduce interference. Even so, interference between co-residents exists, because the scheduler does not account for the sensitivity applications have to resource contention.

The training set consists of 120 diverse applications that include webservers, various analytics algorithms and datasets, and several key-value stores and databases. The training set is selected to provide sufficient coverage of the space of resource characteristics (compute, memory, storage, and network bandwidth). This enables Bolt to match any resource profile to information from the training set, even if the new application has not been previously seen. Increasing the training set size further did not improve detection accuracy. Finally, note that there is no overlap between training and testing sets in terms of algorithms, datasets, and input loads.

Table 1 shows Bolt's detection accuracy, per application class, and aggregate. We signal a detection as correct if Bolt correctly identifies the framework (such as Hadoop) or service (such as memcached) that the application uses, as well as the algorithm (such as SVM on Hadoop) and user-load characteristics (such as read- or write-heavy load). We do not currently examine more detailed features, such as the distribution of individual query types. Bolt correctly identifies 87 percent of jobs, and, for certain application classes such as databases and analytics, the accuracy exceeds 90 percent. Misclassified jobs are typically identified as workloads with the same or similar critical resources.

We also analyzed Bolt's behavior in the presence of multiple co-residents and quantified how many iterations it takes for Bolt to correctly detect all victims. The left side of Figure 2 shows accuracy as a function of the number of victims per machine. When the number of co-residents is less than two, accuracy exceeds 95 percent. As the number increases, accuracy drops, because it becomes progressively more difficult to differentiate between the contention caused by each workload. When there are five co-scheduled applications, accuracy is 67 percent. Interestingly, accuracy is higher for four than for three applications, because, with four workloads, the probability of Bolt sharing a core with a victim and thus getting an accurate measurement of core pressure is higher. While aggressive multi-tenancy affects accuracy, large numbers of co-residents in public clouds are unlikely in practice.

Table 1. Bolt's detection accuracy in the controlled experiment with the LL scheduler.

Application	**Detection Accuracy (%)**
Memcached	78%
Hadoop	92%
Spark	85%
Cassandra	90%
Speccpu2006	84%
Aggregate of all applications	*87%*

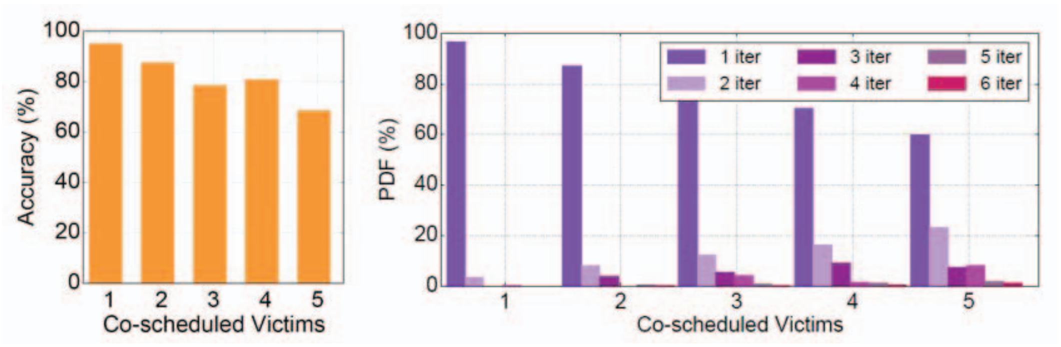


Figure 2. (Left) detection accuracy and (right) probability distribution function (PDF) of iterations required for detection as a function of the number of co-runners.

We now examine to what extent the number of co-scheduled victims affects the number of iterations needed for detection. We signal detection complete once all victims have been correctly identified. As seen on the right side of Figure 2, when there is a single victim in a server, in the vast majority of cases, a single iteration of profiling and data mining is sufficient to detect and label it. The number of iterations goes up as more victims share a single physical node. For example, for three co-scheduled victims, in 22 percent of cases, two or more iterations were needed for correct detection. In all cases, including scenarios with five co-scheduled victims, six iterations were enough to complete all victim detection.

Finally, we examine how design decisions in Bolt affect detection. The left side of Figure 3 shows how accuracy changes as profiling frequency decreases. For profiling intervals beyond 30 seconds, accuracy drops rapidly. If profiling only occurs every five minutes, almost half the victims are incorrectly identified. Note that if workloads are long-running and mostly stable, profiling can be less frequent without such an impact on accuracy. The middle of Figure 3 shows accuracy as a function of the adversarial VM's size. We examine sizes offered as on-demand instances by EC2. If the adversary has fewer than four vCPUs, its resources are insufficient to create enough contention to capture the co-residents' pressure. Accuracy continues to grow for larger sizes; however, the larger the adversarial instance is, the less likely it will be co-scheduled with other VMs, nullifying the value of Bolt. Finally, the right side of Figure 3 shows accuracy as a function of the number of micro-benchmarks used for profiling. A single benchmark is not sufficient to fingerprint the characteristics of a workload; however, using more than three benchmarks has diminishing returns in accuracy. Unless otherwise specified, we use 20-second profiling intervals, four-vCPU adversarial VMs, and two micro-benchmarks for initial profiling.

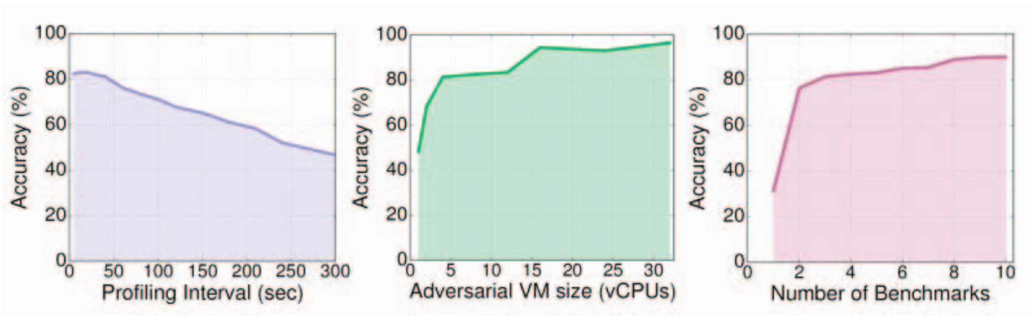


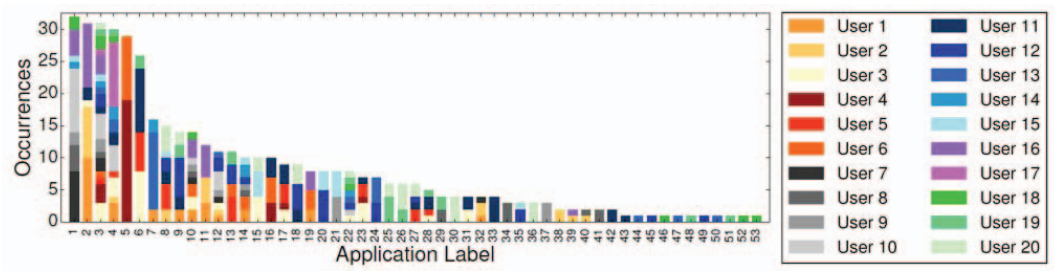
Figure 3. Sensitivity to (left) profiling frequency, (middle) adversarial VM size, and (right) profiling benchmarks.

USER STUDY

We also evaluated Bolt "in the wild" through a 20-user study at Stanford University and Cornell University. We gave users access to a 200-server cluster on Amazon EC2 and allowed them to run any application (without disclosing any information about the type of application to us). Users were instructed to avoid consuming disproportionate resources and to pin their applications to compute cores to avoid interference from the OS scheduler's decisions. They were also free to select specific servers themselves or to allow an LL scheduler to place their jobs on the machines with the most available resources.

Once users started launching jobs in the cluster, we instantiated Bolt on each machine and activated it periodically to detect the type, characteristics, and number of co-scheduled applications. The experiment lasted approximately four hours, after which all instances were terminated. Figure 4 shows the PDF of application types the users launched, provided by the users after the completion of the experiment. Different colors correspond to different users. A total of 436 jobs were submitted to the cluster. The application mix includes analytics jobs in Hadoop and Spark, scientific benchmarks (Parsec and Bioparallel), hardware synthesis tools (Cadence and VivadoHLS), multicore (zsim) and n-body simulations, email clients, web browsers, webservers (http), music and video streaming services, and databases. Each job may take one or more vCPUs. Even though Figure 4 groups applications by framework, the individual logic and datasets differ between jobs in the same framework. Note that we have not updated the training set for this study; it is the same set of applications used in the validation study described previously.

The upper-left image in Figure 5 shows the number of applications that were correctly identified by name across all categories. In total, 277 jobs were correctly labeled. As expected, Bolt could not assign a label to application types it had not seen before, such as email clients and image editing applications. However, even when Bolt could not label an application, it could still identify the resources it is sensitive to, as shown in the upper-right image of Figure 5. For 385 out of 436 jobs, Bolt correctly identified the job's resource characteristics. For performance attacks, such as the ones described below, this information is sufficient to dramatically degrade the performance of a victim application. Finally, the bottom-left image of Figure 5 shows the number of co-scheduled applications per instance throughout the duration of the experiment. The bottom 14 instances remained unused; for the remaining 186 instances, the number of active applications ranged from 1 to 6, with each job taking one or more vCPUs. The majority of misclassified applications corresponded to instances with five or more concurrently active jobs.



1:hadoop	7:video	13:spec	19:scala	25:ppt	31:bioparallel	37:html	43:oProf	49:rm
2:spark	8:latex	14:matlab	20:php	26:OS img	32:storm	38:Cassandra	44:download LF	50:skype
3:email	9:MLPython	15:mysql	21:postgres	27:pdfview	33:cpu_burn	39:mongoDB	45:rsync	51:zipkin
4:browser	10:make	16:vivado	22:music Stream	28:scons	34:audacity	40:mkdir	46:ping	52:graphX
5:cadence	11:mem$	17:parsec	23:minebench	29:du –h	35:javascript	41:cp/mv	47:photoShop	53:ix
6: zsim	12:http server	18:vim	24:n-body sim	30: cgroup	36:create VM	42:sirius	48:ssh	

Figure 4. PDF of applications launched in the cloud study, per user.

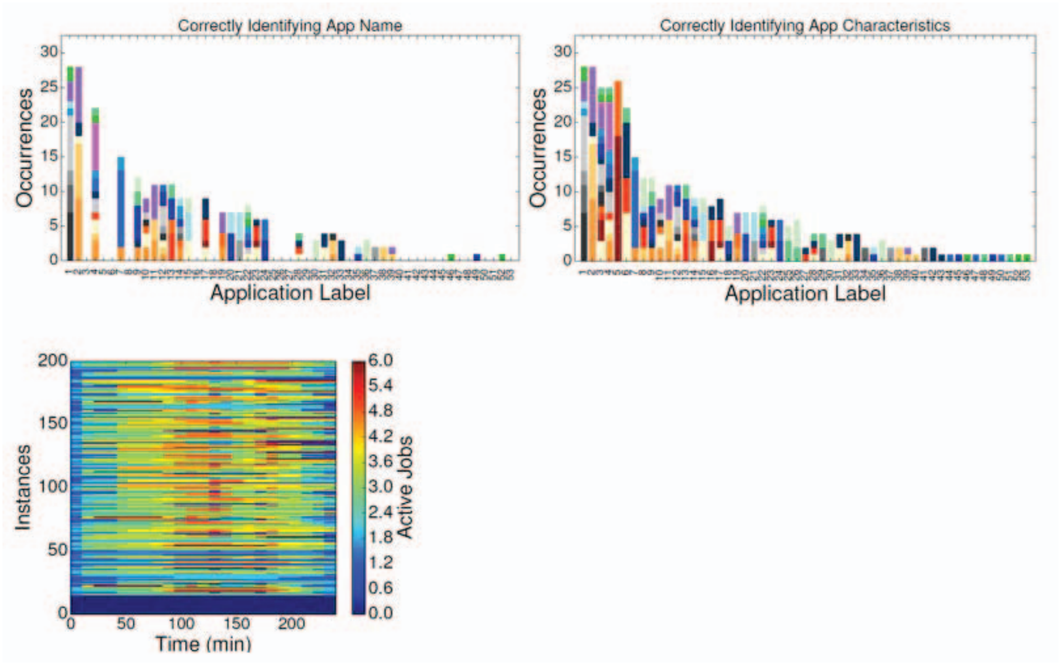


Figure 5. Detection accuracy of (upper-left) application labels and (upper-right) resource characteristics. (Bottom-left) the number of jobs per instance.

BOLT'S ATTACKS

Obtaining the information on the type and characteristics of victim applications enables Bolt to launch several performance attacks, including DoS attacks, RFAs, and VM pinpointing attacks. Knowing which resources the victim is sensitive to enables DoS attacks and RFAs even if the victim could not be labeled. However, given that adversaries often target a specific victim with these attacks, using Bolt to verify that the characteristics of a co-scheduled application match the target victim's behavior increases the effectiveness and cost-efficiency of the performance attack.

Additionally, Bolt makes these attacks very hard to detect. For example, a traditional network or host-based DoS attack aims to saturate a server's resources, preventing the victim from doing useful work. Instead, Bolt uses its knowledge of the resources a victim is sensitive to to inject interference in them, hurting the victim's performance without needing to reach CPU saturation. Not only does this translate to dramatic performance degradation for the victim, but it also makes the adversary hard to detect by the cloud provider. Across the 108 applications of our controlled experiment, Bolt degraded performance by up to 9.8X in execution time, and by 8-140X in tail latency.

We now examine the impact of the DoS attack on utilization for the case of a specific victim. If the DoS attack translates to resource saturation, there is a high probability that it will be detected by the cloud provider's monitoring mechanisms and the victim (or adversary) will be migrated to a new machine. The experimental cluster supports live migration. If CPU utilization (sampled every 1 s) exceeds 70 percent, the victim is migrated to an unloaded host. Figure 6 compares the tail latency and CPU utilization with Bolt to that of a naïve DoS attack that saturates the CPU through a compute-intensive kernel. We focus on a single victim VM running memcached. The overhead of migration (time between initiating migration and latency returning to normal levels) for this VM is 8 s. Performance degradation is similar for both systems until t = 80 s, at which point the victim is migrated to a new host due to the compute-intensive kernel causing utilization to exceed 70 percent. While, during migration, performance continues to degrade, once the VM resumes in the new server, latency returns to nominal levels. In contrast, Bolt keeps utilization low and impacts the victim's performance beyond t = 80 s.

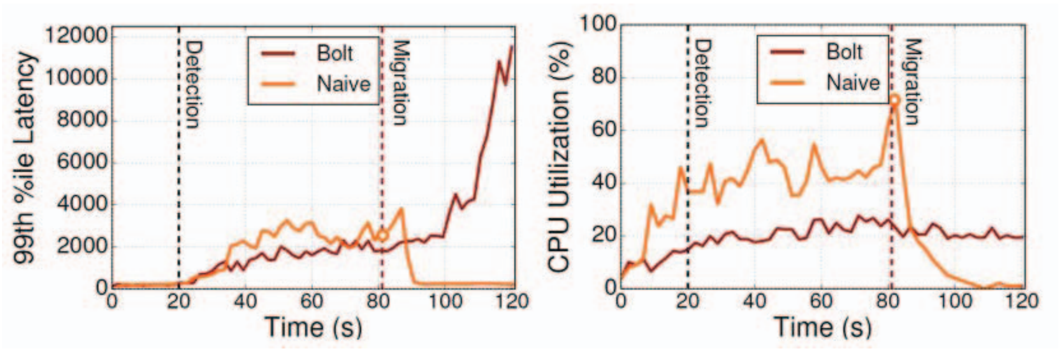


Figure 6. Latency and utilization with Bolt and a naïve DoS attack that saturates CPU resources.

THE ROLE OF RESOURCE ISOLATION

Because interference is at the heart of Bolt's detection methodology, isolation mechanisms that mitigate contention should reduce its accuracy. We first evaluate to what extent existing isolation techniques mitigate security vulnerabilities, and then we highlight trade-offs between security, performance, and utilization. We evaluate four resource-specific isolation techniques and three settings for OS-level isolation mechanisms: baremetal, containerized, and virtualized. The baremetal system does not employ any OS-level isolation. The containerized setup uses Linux containers (lxc) and assigns cores to applications through cpuset cgroups. Both containers and

VMs constrain memory capacity. Baremetal experiments do not enforce memory capacity allocations, and the Linux scheduler is free to float applications across cores.

The first resource-specific isolation techniques is thread pinning to physical cores to constrain interference from scheduling actions, such as context switching. The number of cores allocated to an application can change dynamically and is limited by how fast Linux can migrate tasks, typically in the tens of milliseconds.

For network isolation, we use the outbound network bandwidth partitioning in Linux's traffic control through the qdisc scheduler with hierarchical token bucket queueing discipline. For DRAM bandwidth isolation, there is no commercially available partitioning mechanism. To enforce bandwidth isolation, we use the following approach: We monitor the DRAM bandwidth usage of each application through performance counters[7] and modify the scheduler to only collocate jobs on machines that can accommodate their aggregate peak memory bandwidth requirements. Finally, for last level cache (LLC) isolation, we use the cache allocation technology (CAT) available in recent Intel chips. CAT partitions the LLC in ways that, in a highly associative cache, allows for non-overlapping partitions at the granularity of a few percent of the LLC capacity. Each co-resident is allocated one partition configured to its current capacity requirements. Partitions can be resized at runtime by reprogramming specific low-level model-specific registers (MSRs); changes take effect after a few milliseconds.

Figure 7 shows the impact of isolation techniques on Bolt's detection accuracy when adding one isolation mechanism at a time. As expected, when no isolation is used, baremetal allows for a significantly higher detection accuracy than container- and VM-based systems, mostly due to the latter constraining core and memory capacity usage. As a result, introducing thread pinning benefits baremetal the most, because it reduces core contention. It also benefits container- and VM-based setups to a lesser degree, by eliminating unpredictability introduced by the Linux scheduler, such as context switching. The dominant resource of each application determines which isolation technique benefits it the most. Thread pinning mostly benefits workloads bound by on-chip resources, such as L1/L2 caches and cores. Adding network bandwidth partitioning lowers detection accuracy for all three settings almost equally. It primarily benefits network-bound workloads, for which network interference conveys the most information for detection. Memory bandwidth isolation further reduces accuracy by 10 percent on average, benefiting jobs dominated by DRAM traffic. Finally, cache partitioning causes the most dramatic reduction in accuracy, especially for LLC-bound workloads. We attribute this to the importance cache pressure has as a detection signal. The number of co-residents also affects the extent to which isolation helps. The more co-scheduled applications exist per machine, the more isolation techniques degrade accuracy, as they make distinguishing between co-residents harder.

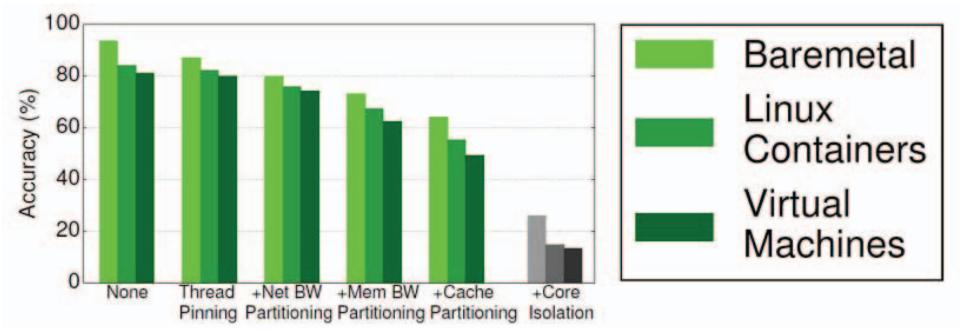


Figure 7. Detection accuracy with isolation techniques.

Unfortunately, even when all isolation techniques are used, accuracy is still 50 percent for two reasons:

- Current techniques are not fine-grained enough to allow strict and scalable isolation, and
- Core resources (L1/L2 caches and CPU) are prone to interference due to contending hyper-threads.[8]

To evaluate the latter hypothesis, we modify the scheduler such that hyper-threads of different jobs are never scheduled on the same physical core (for example, if a job needs seven vCPUs, it is allocated four dedicated physical cores). The gray bars in Figure 7 show the detection accuracy. Baremetal instances still allow certain applications to be detected, but for containerized and virtualized settings, accuracy drops to 14 percent. The remaining accuracy corresponds to disk-heavy workloads. Improving security, however, comes at an average performance penalty of 34 percent in execution time, as threads of the same job are forced to contend with one another. Alternatively, users can overprovision their resource reservations to avoid degradation, which results in a 45-percent drop in utilization. This means that the cloud provider cannot leverage CPU idleness to share machines, decreasing the cost benefits of cloud computing. Note that enforcing core isolation alone is also not sufficient, as it allows a detection accuracy of 46 percent.

LESSONS FROM BOLT

There are three contributions of Bolt that inform the way we design and manage future multi-tenant cloud systems.

First, Bolt showed that there is a current and readily exploitable security threat in modern cloud providers. It quantified the ability of an adversary to obtain confidential information about the type and characteristics of a victim application with minimal profiling, and to do so completely transparently for both the victim and cloud operator. It also showed that the resulting performance attacks are nowhere near as costly or easy to detect as traditional DoS attacks, only requiring co-residency with the victim on a single node to cause equivalent performance degradations. The feasibility and practicality of such attacks in modern cloud infrastructures hurts and delays cloud adoption, especially for interactive applications that are particularly sensitive to unpredictable performance.

Second, the analysis of isolation techniques above shows that existing hardware and software techniques are insufficient to mitigate security vulnerabilities, and techniques that provide reasonable security guarantees sacrifice performance or cost-efficiency, by resulting in low utilization. This highlights the need for new fine-grained and coordinated isolation techniques that guarantee security at high resource utilization. As more cores are integrated per server and the degree of multi-tenancy increases, the need for scalable isolation becomes more pronounced. This analysis also provides the relative importance of isolating different resources, given the fact that some leak much more substantial information about a victim's behavior than others. This resource ranking can help cloud operators prioritize which isolation mechanisms to integrate to their systems.

Bolt shows the value big data can offer in improving the design and management of platforms whose scale makes empirical optimizations impractical.

Third, despite the malicious way in which data mining is used here, Bolt shows the value big data can offer in improving the design and management of platforms whose scale makes empirical optimizations impractical. The application detection process Bolt relies on would have been impossible to perform manually, even if we assumed that the adversary can remain co-scheduled with the same victim for long periods of time.

CONCLUSION

As cloud systems scale in size, number, and complexity, the goals of performance, efficiency, and security often clash with one another. Leveraging practical data mining techniques to quickly and accurately uncover and resolve vulnerabilities or inefficiencies in large-scale systems is a promising approach to reconcile these often-conflicting objectives.

SIDEBAR: RESEARCH RELATED TO BOLT

Performance unpredictability is a well-studied problem in public clouds that stems from platform heterogeneity, resource interference, software bugs, and load variation.[2,3,9] A lot of recent work has proposed isolation techniques to reduce unpredictability by eliminating interference.[7,8] With Bolt, we show that unpredictability due to interference also hides security vulnerabilities, because it enables an adversary to extract information about an application's type and characteristics. Below, we discuss related work on cloud vulnerabilities, such as VM placement detection, distributed denial of service (DDoS) attacks, and side-channel attacks.

VM Co-Residency Detection

Cloud multi-tenancy has fostered attacks that aim to locate a target VM in a public cloud. Ristenpart et al.[4] showed that the IP machine naming conventions of cloud providers allowed adversarial users to narrow down where a victim VM resided in a cluster. This study resulted, in part, in cloud providers changing their naming conventions, reducing the effectiveness of network topology-based co-residency attacks. Bolt does not rely on knowing the cloud's network topology or host IPs, making it resilient to mitigation techniques such as Amazon's Virtual Private Clouds (VPCs).

Performance Attacks

Once a victim application is located, an adversary can negatively affect its performance. DDoS attacks have increased in number and impact over the last few years. This has generated a lot of interest in detection and prevention techniques.[10] More aggressive than DoS attacks, RFAs hurt an application's performance, while additionally forcing it to yield its resources to the adversary.[11] While these attacks are effective, they require significant compute and network resources and are prone to defenses, such as live VM migration, that cloud providers employ. In contrast, Bolt launches host-based attacks on the same machine as the victim that keep resource utilization moderate, therefore evading detection.

Side-Channel Attacks

There are also attacks that attempt to extract confidential information from co-scheduled applications, such as private keys. Zhang et al.[5] proposed a system that launches side-channel attacks in a virtualized environment and cross-tenant side-channel attacks in platform as a service (PaaS) clouds. On the defense side, Wang et al.[12] proposed a system for intrusion detection in cloud settings. Bolt does not rely on the accurate microarchitectural event measurements many side-channel attacks need, and it is therefore resilient to techniques that limit the fidelity of performance monitoring to thwart information leakage in side-channel attacks.[13]

REFERENCES

1. L. Barroso and U. Hoelzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, MC Publishers, 2009.
2. C. Delimitrou and C. Kozyrakis, "Quasar: Resource-Efficient and QoS-Aware Cluster Management," *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.
3. C. Delimitrou and C. Kozyrakis, "Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters," *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.
4. T. Ristenpart et al., "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," *ACM Conference on Computer and Communications Security (CCS)*, 2009.

5. Y. Zhang et al., “Cross-vm side channels and their use to extract private keys,” *ACM Conference on Computer and Communications Security (CCS)*, 2012.
6. C. Delimitrou and C. Kozyrakis, “Bolt: I Know What You Did Last Summer... In The Cloud,” *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.
7. D. Lo et al., “Heracles: Improving resource efficiency at scale,” *International Symposium on Computer Architecture (ISCA)*, 2015.
8. D. Sanchez and C. Kozyrakis, “Vantage: Scalable and Efficient Fine-Grain Cache Partitioning,” *International Symposium in Computer Architecture (ISCA-38)*, 2011.
9. C. Delimitrou and C. Kozyrakis, “HCloud: Resource-Efficient Provisioning in Shared Cloud Systems,” *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.
10. M. Darwish, A. Ouda, and L.F. Capretz, “Cloud-based DDoS attacks and defenses,” *i-Society*, 2013.
11. V. Varadarajan et al., “Resource-freeing attacks: Improve your cloud performance (at your neighbor’s expense),” *ACM Conference on Computer and Communications Security (CCS)*, 2012.
12. H. Wang, H. Zhou, and C. Wang, “Virtual machine-based intrusion detection system framework in cloud computing environment,” *Journal of Computers*, 2012.
13. R. Martin, J. Demme, and S. Sethumadhavan, “Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks,” *International Symposium on Computer Architecture (ISCA)*, 2013.

ABOUT THE AUTHORS

Christina Delimitrou is an assistant professor in the Departments of Electrical and Computer Engineering and Computer Science at Cornell University, where she works on computer architecture and distributed systems. Her research interests include resource-efficient datacenters, scheduling and resource management with quality-of-service guarantees, disaggregated cloud architectures, emerging cloud and IoT application models, and cloud security. Delimitrou has a PhD in electrical engineering from Stanford University. She is a member of the IEEE and ACM. Contact her at delimitrou@cornell.edu.

Christos Kozyrakis is a professor in the Departments of Electrical Engineering and Computer Science at Stanford University, where he investigates hardware architectures, system software, and programming models for systems ranging from cell phones to warehouse-scale datacenters. His research interests include resource-efficient cloud computing, energy-efficient computing and memory systems for emerging workloads, and scalable operating systems. Kozyrakis has a PhD in computer science from the University of California, Berkeley. He is a Fellow of the IEEE and ACM. Contact him at kozyraki@stanford.edu.