CloudFlow: Cloud-wide policy enforcement using fast VM introspection

Mirza Basim Baig, Connor Fitzsimons, Suryanarayanan Balasubramanian,
Radu Sion and Donald Porter
Computer Science, Stony Brook University
{mbaig,crfitzsimons,sbalasubrama,sion,porter}@cs.stonybrook.edu

***Abstract*—Increasingly government and commercial enterprises are considering cloud adoption. Clouds share hardware resources across a number of software virtual machines. While this carries the promise of increasingly efficient hardware utilization, it also comes with inherent side-channel vulnerabilities which can be used to mount attacks that cause undesired information leakage.**

This is especially important in infrastructures shared by multiple regulatory constrained, security-disjoint principals. A classic example is a financial company with multiple departments under the incidence of Chinese Wall style regulatory policies designed to curtail illicit gains and conflicts.

And, while at the individual node level, the prevention of side channels is hard without sacrificing efficiency, a unique mitigation opportunity exists cloud-wide. By reactively enforcing information flow invariants through run-time introspection and on-demand virtual machine migration we can prevent undesired node-level side-channels with minimal performance overhead.

In this paper we introduce CloudFlow – an information flow control mechanism for OpenStack that deploys a new, fast virtual machine introspection mechanism (orders of magnitude faster than previous approaches) to efficiently and transparently enable policy enforcement cloud-wide.

CloudFlow mitigates undesirable side-channels, provides protection against undesirable information leaks and forms a powerful tool enabling information flow policy enforcement at cloud scope. Additionally, CloudFlow has potential uses for cloud management and resource-efficient virtual machine scheduling.

I. Introduction

The flexibility and efficiency of the virtualization based cloud computing makes it extremely attractive for in-house deployments where companies set up cloud based data centers and share hardware resources across all their users.

Nevertheless, clouds come with a number of new security issues, a primary one amongst them being the inherent existence of side-channels due to resource-sharing.

Virtual machines (VMs) sharing the same hardware are vulnerable to malicious side-channel attacks [25, 34, 37, 38, 42] making the enforcement of proper cloud-wide information flow control difficult. Yet, this is a major requirement for (in-house) clouds that may be hosting workload from different units which should be mutually isolated for regulatory and security reasons.

For example, information flow control is an essential part of enforcing regulatory compliance required in e.g., heavily regulated financial companies which need to enforce Chinese wall policies to segregate resources used by different internal departments and provide proof that no conflicts of interest exist as a result of unwanted information leaks.

History has shown that self-regulation does not always work [2] and as a result laws have been promulgated [8] requiring e.g., the brokerage and investment arms of the same financial organization to be separated via a Chinese Wall. Similar regulation governs interactions in healthcare enterprises, financial ratings companies etc.

Intra-node and limited inter-node information flow control mechanisms have been researched for decades and have resulted in several practical systems such as Histar [39] and SELinux [12] which aim to provide general MLS and information flow control mechanisms at OS level.

However, since node-level side-channels cannot be eliminated efficiently when resources are shared across multiple co-resident VMs, it is important to explore whether and how cloud-wide information flow invariants can be enforced.

Unfortunately, the explosive emergence of clouds has left its adopters without access to information flow control tools that work at cloud scope, and businesses are forced to deploy inefficient, ad-hoc solutions such as manual segregation of internal networks and hosts to solve their regulatory compliance information flow control requirements. Overall, private cloud technology is missing a security platform that can translate high level information flow control policies, such as role based user and workflow isolation, into micro security decisions that are enforced infrastructure-wide at runtime.

Initial work (Home Alone [41]) has tackled certain issues of side-channel prevention in public clouds by allowing a tenant to verify its VMs' exclusive use of a physical machine.

In this paper we go one step further and design a cloud-wide, automatic information flow control mechanism. CloudFlow is a suite of lightweight modules designed for OpenStack [6] that deploy fast hypervisor-level VM introspection, management and migration to efficiently and transparently enforce cloud-wide information flow control invariants. The main philosophy behind CloudFlow is to not require any changes to the hosted guest VMs or applications.

The main contributions of this work are as follows. Firstly, we provide a new introspection mechanism for KVM-QEMU [15] that allows for fast asynchronous VM introspection at the hypervisor level. Our technique is several orders of magnitude faster than previous libraries such as libvirt [29] and libVMI [30]. Secondly we design a cloud-wide information flow

control layer for OpenStack [6] that leverages the introspection mechanisms to enforce best-effort real-time information flow control.

II. Background and Model

a) **Regulatory compliance & Information flow control:** People do not trust institutions when they believe that the appropriate policies to deter abuse are lacking or not being enforced. When the public trust is threatened [2], often new regulations and policies are put into place to restore trust. Over 10,000 IT-impacting regulations exist in the US alone, including Sarbanes-Oxley Act [8], Health Insurance Portability and Accountability Act [4], Gramm-Leach-Bliley Act [5], Federal Information Security Management Act [3], Securities and Exchange Commission (SEC) rule 17a-4 [11], the e-Government Act [9], and the Patriot Act [10].

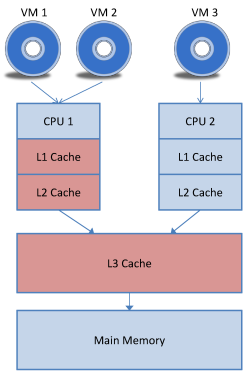
The core issue that many of these laws ultimately address is the need for fair and transparent control of sensitive information. Accordingly, information flow control is an essential necessity within any large organization. However, with scale come increasing difficulties in enforcing information flow control, and illicit accesses can go unmonitored.

b) **MLS and Chinese Wall policies:** Of particular importance in regulatory contexts are Chinese Wall policies that institutions need to enforce to mitigate existing and potential conflicts of interest. A *Chinese Wall* is a barrier placed across information flow within an organization that aims to segregate employees who have access to sensitive information from those who may use this information for personal gain. Traditionally this is done by isolating personnel from each other and monitoring information access and sharing. As we will see, CloudFlow can naturally enforce Chinese wall policies by dynamic inter-VM segregation for applications running on behalf of principals under regulatory constraints.

Similarly, in intelligence and defense scenarios, information flow control is subject to *Multilevel security* (MLS) policies. MLS systems process information that exists at differing, often incompatible, security or *classification levels* and provide access control mechanisms for users at different security clearances and needs-to-know, preventing illegitimate access to classified information. One popular example is the well known Bell-Lapadula confidentiality policy model [1]. Traditionally, Bell-Lapadula uses four different classification levels, from Top Secret down to Unclassified. Information flow is controlled according to "no reads up, no writes down" rules. CloudFlow can also naturally enforce MLS policies as will be shown later.

There has been previous work on network-centric cloud-wide policy enforcement [16, 19, 20, 32, 36]. Policies are usually defined in order to attain better segregation amongst nodes of different computational and storage needs and provide better load balancing or scheduling algorithms. Work that most closely resembles CloudFlow includes a result that proposes stronger VM isolation mechanisms [40]. Such mechanisms rely mainly on providing rudimentary MLS via hardwired network isolation.

c) **Side Channels:** In a *side channel attack* information is gained from a system using a resource that is shared between an unsuspecting victim and a perpetrator. Examples of such shared resources include power conduits, discs, shared memory hierarchies etc. all of which can be used to infer potentially sensitive information illegitimately.



An especially important and interesting class of side channels arises due to improper VM isolation. This derives from the inherent resource sharing design that comes with efficient infrastructures. For example, clouds usually multiplex multiple VMs on the same underlying hardware by assigning a virtual CPU (VCPU) to each VM. The number of VCPUs can be greater than the physical CPUs and hence, in the case of load balancing a large number of VMs, different VCPUs end up getting mapped to the same physical CPU. Two VMs on the same physical CPU end up sharing all levels of the memory hierarchy. This can lead to information leakage e.g., via shared caches [34, 37]. Even though the two VMs share no explicit state, the CPU cache is a shared resource impacted by the internal state of both VMs (see Figure). This introduces a side channel that potentially breaks desired information flow control invariants. Numerous other side channels have been discovered in a variety of settings and have been studied extensively [25, 34, 37, 38]. Associated exploits have been designed and implemented to extract private information such as RSA and AES keys [34, 37, 42] via side channels from unsuspecting victims.

While preventing individual side channels at OS and hypervisor level is difficult and arguably impossible to achieve in efficient resource sharing systems, we believe a unique mitigation opportunity exists at cloud level. By reactively enforcing information flow control invariants through runtime introspection and on-demand VM migration the effects of existing side channels can be mitigated. However we acknowledge the fact that any system that is reactive in nature can be subject to *residual side channels* i.e. whenever a policy is enforced at runtime the attacker has the opportunity to learn information about the policy being enforced. This is not an issue in the regulatory compliance scenario under focus in this paper as policies themselves are not classified information and do not need to be protected from side channel attacks.

d) **VM Introspection:** Introspection was first suggested in the seminal work of Garfield and Rosenblum [23]. Two types of introspection mechanisms exist: those requiring changes to the guest VM or target applications, and those that work externally at the hypervisor level. CloudFlow aims for maximum transparency and minimizing the impact on guest OSes and applications and thus uses external introspection.

Introspection has been extremely popular in the recent past [14, 26–28, 30, 31, 35]. Previous introspection mechanisms have been aimed towards solving problems such as intrusion detection [33], malware analysis [21] and forensics.

External introspection approaches however are inherently at a disadvantage due to having an outsider view and remain

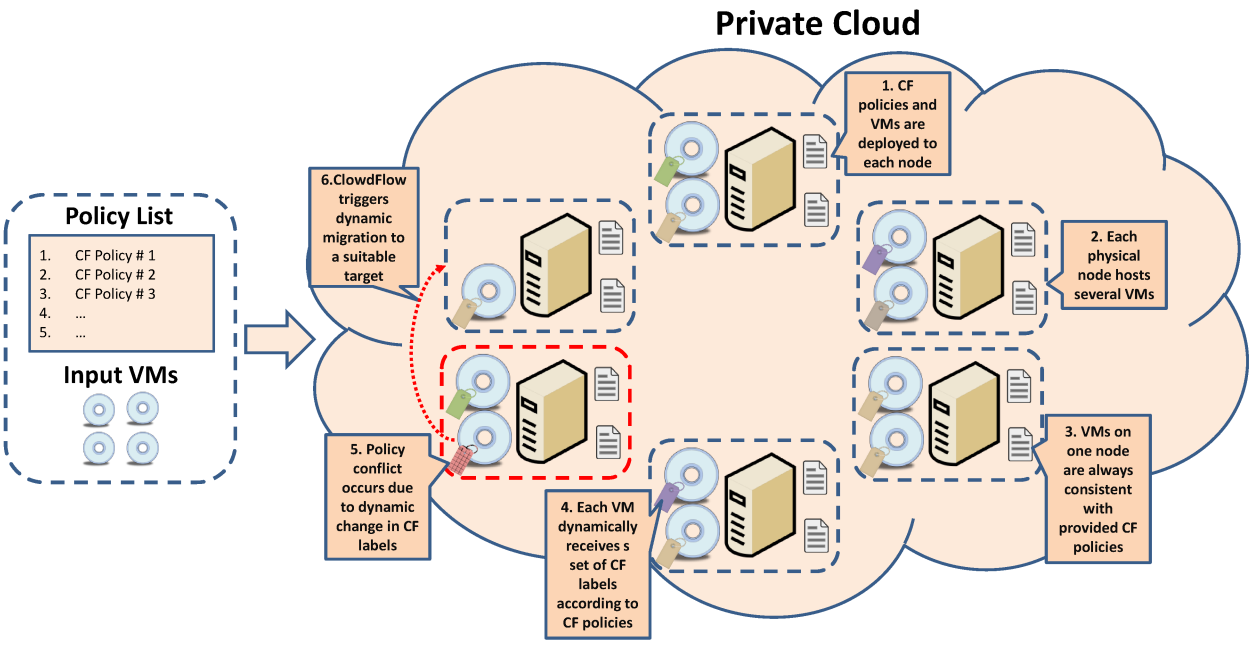


Fig. 1. Administrators from different departments provide CloudFlow with VM images and desired security policies. CloudFlow (CF) deploys the images to the cloud and propagates CF policies automatically to relevant nodes. The policies are then used to dynamically deduce CF labels for VMs. Policies are enforced in real-time using VM introspection and call-backs to the management module(e.g. migration requests). The figure shows a migration in progress.

prone to the *semantic gap* problem [17, 18], the difficulty of mapping between low-level hardware events and higher-level system abstractions. Further, in practice, interpreting the memory of a running guest OS can be error prone, and subject to attacks [13].

In recent years there has been much effort to reduce this gap and gain meaningful, detailed information even in the case of non-cooperating, adversarial guest VMs [22, 24]. This has led to the construction of hybrid approaches that install an agent inside the monitored guest as well, to provide a view that is richer in semantics.

e) **Clouds:** Cloud computing enables the outsourcing of computation and storage needs to an infrastructure managed by a third party (public cloud provider, or in-house company-wide cloud/IT department). Clouds today are mostly virtualization based i.e. they use VMs as the basic unit for leasing out computation. They usually provide a management front-end which can be used as an input point to deploy and manage workload-personalized VMs which are run in a data center composed of physical nodes that provide computation and storage, and can host multiple VMs simultaneously.

CloudFlow is based on the OpenStack [6] framework. OpenStack offers a variety of services including computing and storage nodes, and works with a wide range of hypervisors. OpenStack is research friendly and provides open source code which facilitated our design. In addition, we have picked the popular KVM-QEMU as our hypervisor of choice. Nevertheless we note that CloudFlow is hypervisor agnostic and it can be easily ported to other cloud platforms.

f) **Deployment Model:** The key idea behind CloudFlow is to enable hypervisors and the cloud management layers to dynamically enforce cloud-wide information flow policies that are based on guest VM state introspected at runtime. For example, in the case of Bell-Lapadula/MLS enforcement, a policy may specify that no VM running applications classified/labeled as ”confidential” can be hosted on the same physical node as a VM running an application handling ”top secret” labeled data. No application and guest OS changes should be required.

Naturally, the granularity at which guest state can be monitored via introspection can vary vastly depending on the depth of data extracted from the monitored kernel. For illustration purposes, the current version of CloudFlow allows policies containing references to SELinux labels of guest-hosted processes. The set of labels is currently derived from the typing system put in place by SELinux inside each guest.

g) **Threat Model & Assumptions:** Of major concern are insiders who will benefit from cross-VM side channels. Our main insight is that while at individual node level, the prevention of side channels is tantamount to sacrificing efficiency, at cloud level we can reactively prevent undesired side channels by intelligent on-demand policy-enforcing VM placement, without a significant performance hit.

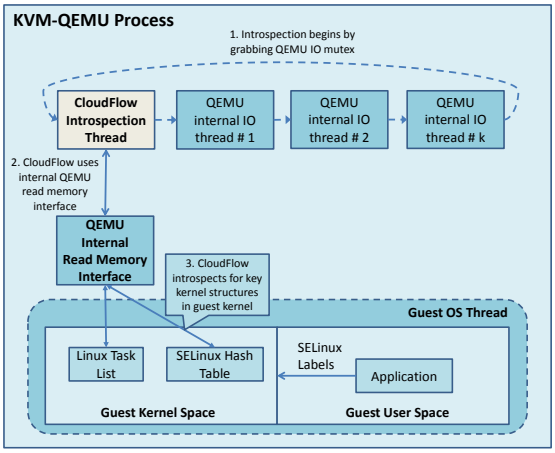


Fig. 2. CloudFlow deploys a fast introspection thread that uses KVM-QEMU internals directly to extract key kernel structures and infer information about tasks running inside the guest OS.

We trust the hypervisor, cloud administrators and policy designers. All other users are untrustworthy.

Typically attackers will deploy a a malicious VM through which they will attempt to exploit a known hardware side channel and extract sensitive information from a unsuspecting target VM. We assume that the adversary is not in control of the entire underlying hardware and is unable to bypass the hypervisor isolation.

Further, while we do not fully trust the guest OS and its applications, we assume that guest kernels do not aim to subvert the introspection mechanism.

Nevertheless, we note that recent work [13] has shown it is possible to deceive hypervisor based introspection mechanisms via an attack on the guest kernel. This is why ideally the guest OS should be entirely untrusted.

For this reason we designed a framework that can be ultimately independent of guest support. Additionally, we note that because CloudFlow does not modify the guest OS it does not need constant maintenance upon every kernel version update. CloudFlow can run on multiple kernel versions and different operating systems by simply changing how the introspection is performed externally. Finally, CloudFlow allows us to set a base for the policy management and information flow control that remains agnostic of how the guest state is actually acquired.

In ongoing work we are developing deception-robust introspection mechanisms that can handle fully malicious guest kernels.

III. Architecture & Implementation

CloudFlow is a modular system composed of several asynchronously connected components. It provides a cloud-wide enforcement mechanism, assisting in information flow control via active policy deployment and on demand VM relocation. CloudFlow only requires VM images and policies as input (Figure 1). Policies are propagated to nodes on-demand at runtime.

During execution, each VM has a set of application-specific labels associated with its running tasks. The appearance or disappearance of a guest-hosted application with a given runtime label may trigger a policy-defined action. This action may be null or may result in the relocation or halting of a subset of VMs. The ultimate purpose is reactive defense against any potentially unallowed information flows, as defined by the currently deployed policies.

CloudFlow can naturally enforce different types of information flow policies including Chinese Wall and Bell-Lapadula MLS models. For example in the case MLS policies, the runtime labels can be thought of as MLS classification labels and CloudFlow as a runtime MLS enforcement mechanism.

A. Introspection Module

CloudFlow deploys VM introspection to extract runtime labels from key guest kernel data structures. CloudFlow enforces information flow in real-time which renders existing introspection libraries for KVM-QEMU unusable due to their relatively high latency and low speeds. For these reasons, CloudFlow undertakes VM introspection by hooking into existing KVM-QEMU internals and significantly reducing overheads due to unnecessary levels of indirection added by external libraries.

Introspection is performed by a specialized introspection thread, one instance of which is placed near each monitored VM. The introspection thread runs as part of the KVM-QEMU hypervisor code base [15] and exists inside the KVM-QEMU process. It inserts itself inside the scheduling chain of hardware I/O threads (within KVM-QEMU) that manage hardware I/O virtualization for the hypervisor (Figure 2). This allows the use of KVM-QEMU's internal mechanisms for quick access to guest memory. QEMU uses these mechanisms to manage DMA between I/O threads and the guest operating system however we leverage them for quick introspection. The introspection thread synchronizes with the other threads using existing locking mechanisms. Yet since KVM-QEMU follows a hybrid approach that allows both multi-threading and event based programming, the introspection thread is also aware of internal scheduling demands of KVM-QEMU and yields the I/O lock often enough to allow graceful handling of KVM-QEMU events.

With a hook into the KVM-QEMU's internal guest memory reading interface and a number of synchronization issues resolved, the introspection thread feeds from two key guest kernel data structures: the task list and the SELinux hash table containing context information for all running tasks. The thread extracts this information periodically, monitoring for changes to the running task list and associated SELinux context information. The thread receives a list of target labels of interest (from the policy infrastructure) and continuously searches for them inside the target guest. Once a related change happens inside the guest kernel, the information is propagated from the guest to the policy checking modules

```
<policy-list> = <policy>
              | <policy> <policy-list>

<policy> =  <p> if <label> in <locality>
                then <action> </p>

<label> = {Set of target runtime labels}

<locality> = <co-resident> | <self>

<action> = <migrate> | <halt> | resume
```

Fig. 3. Subset of CloudFlow Policy Language

within a few milliseconds[1] (which constitutes a vulnerability window discussed later).

Also, for extensibility and re-usability, the introspection thread is segregated into two separate components. The first component handles the interfacing with KVM-QEMU internals, while the second component provides the SELinux-aware data extraction mechanisms.

Since only the second component is entangled with CloudFlow particulars, the first component in fact constitutes a new fast introspection interface for KVM-QEMU which will be made public as a QEMU patch.

B. Policy Language Model

As previously mentioned, CloudFlow follows a policy-centric approach. Its policy language provides a way to delineate information flow control decisions in the form of high level policy constructs a subset of which is shown in Figure 3.

For simplicity and efficiency purposes we have decided to keep the CloudFlow language as simple as possible[2] while still serving as a sound illustrative example. Currently the policy language has a simple grammatical model. Each policy list is made up of one or more policies. Each policy expects a label, the locality of the target virtual machine and an appropriate action to perform in case of a positive match for the provided label. Presently, the set of labels is limited to SELinux [12] types. However it must be noted that even though we use SELinux types as labels, our policies are simple in nature and do need expertise to craft as is the case with SELinux policies. We felt there is value in coupling with guest SELinux ecosystems, to allow for a wide range of complex policies, yet not require changes to existing guest logic. We note that the set of labels can be changed arbitrarily along with the underlying OS.

All label symbols form terminal symbols in the policy grammar. The set of localities is formed of two terminal symbols <co-resident> and <self>. As all policies are written from the viewpoint of a particular VM, this part of the policy outlines whether the label should be searched for in the VM itself (self) or in all the VMs co-resident to it (co-resident). Finally the set of actions is formed of three terminal symbols <migrate>, <halt> and <resume>. These symbols are pretty self-explanatory as they outline the action that needs to be taken on the VM that contained the label specified by the current policy.

Consider for example the case when VMs for users Alice and Bob need to be kept physically isolated throughout the cloud infrastructure – the corresponding policy for Bob is illustrated in Figure 4. In this example, processes run by Alice are marked as “Alice” (as well with a number of other labels, including the process name) at runtime by the SELinux subsystem on each guest to facilitate in the introspection process.

Further, we have picked XML as the format of choice for the actual implementation of the policy files. The policy language itself is agnostic to the format used in the implementation. In fact, early versions of CloudFlow were using key-value pairs to implement the policy language. XML has a flexible structure that allows quick extensions with minimal effort. As CloudFlow policies and associated scenarios mature we envision a set of graphical tools to manipulate these cloud-wide policies.

```
<p> if <Alice> in <co-resident>
    then <migrate> </p>
```

Fig. 4. Example policy

C. Management Module

The CloudFlow management module is the component that provides an entry point to the cloud administrator. An administrator can input a VM image along with a list of security policies written by a policy designer. In addition to the usual management tasks, such as deploying and tracking the VMs, the CloudFlow management module does all the housekeeping such as storing and propagating policies. The module is integrated seamlessly in the OpenStack platform [6] and features additional code to handle call-backs from the policy module as well as user interface events.

D. Policy Module (Daemon, Master)

The CloudFlow management module propagates the provided policies to the actual policy enforcement infrastructure. CloudFlow runs a cloud-wide policy master module (PolicyM) and a per-node instance of the policy daemon (PolicyD). PolicyM acts as a master that controls all running instances of PolicyD and keeps track of conflicts across physical hosts. Meanwhile, PolicyD has the job of maintaining policy lists for a given physical node as well as interfacing with the introspection module for each VM.

Whenever a new policy is passed to PolicyD by the management module at VM deployment time, it parses target labels in the policy and passes them on to the appropriate introspection module. It also notifies PolicyM that a new

[1]This is a vast improvement over latency caused by using existing introspection libraries as we show in Section V.

[2]Obviously any arbitrary Turing-complete language could be deployed here

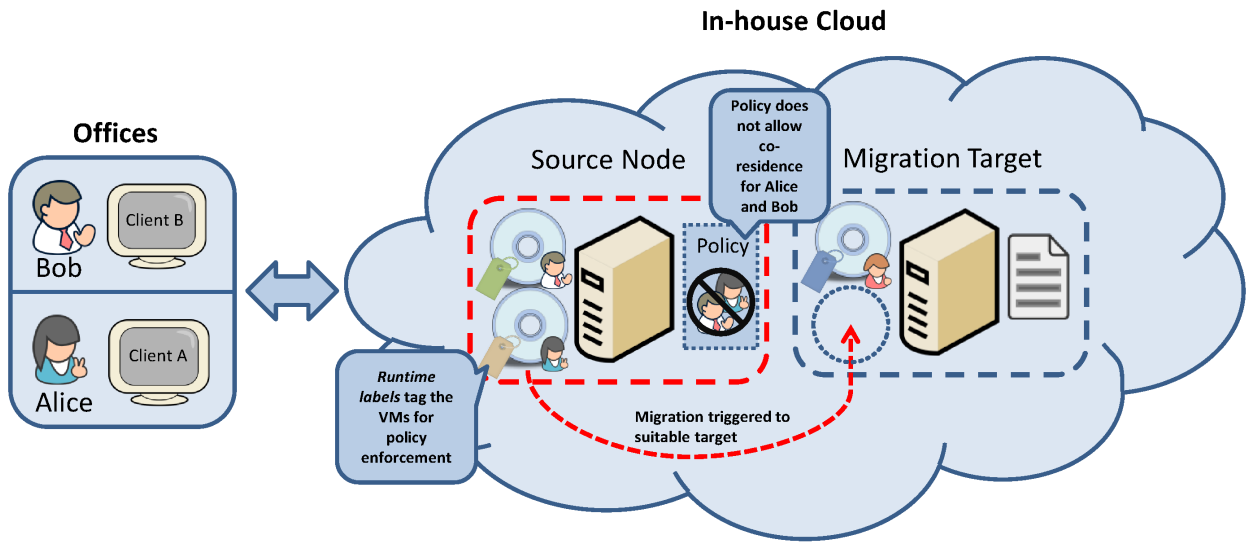


Fig. 6. CloudFlow performs dynamic policy-triggered VM relocation based on input policies. An example policy is one that disallows two users (Alice and Bob) from being co-resident on the same physical node based on *runtime labels*.

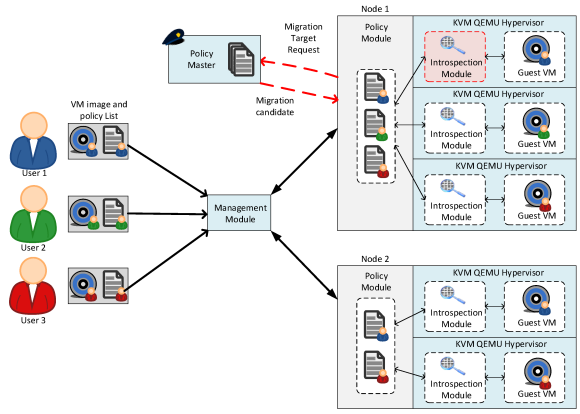


Fig. 5. CloudFlow follows a completely modular design. VM management, policy enforcement and VM introspection are all performed by dedicated modules.

VM has become active in the cloud. Similarly in the case of VM deletion, PolicyM is informed by PolicyD of the change. Additionally, whenever a policy label "matches" a runtime label inside a guest VM, PolicyD gets notified by the corresponding introspection module, in effect "triggering" a policy. PolicyD looks up the required action as specified by this policy. In case of a "halt" action, PolicyD simply issues a halt call-back to the management module. In case of a "migration" it issues a request to PolicyM to search for a potential migration target.

PolicyM maintains high level state for the entire cloud and in case of a migration request from PolicyD, it scouts for a target node that does not introduce additional conflicts. To this end PolicyM executes a reconnaissance phase to find the potential migration target. It queries runtime labels for all deployed VMs from the associated PolicyDs and crosschecks them with deployed policies. This allows for the selection of a migration target that (at least in the immediate future) will not cause further conflicts. PolicyM then reports this host back to PolicyD, which issues the migration request call-back to the management module. The CloudFlow PolicyD and PolicyM were written from scratch in Python and Java respectively.

E. Illustrative Scenario

Here we outline an illustrative scenario (Figure 6). Consider a law firm with two clients A and B, handled by employees Alice and Bob respectively. State and federal laws prevent Alice and Bob from sharing any information due to the possibility of a conflict of interest arising. In this scenario, CloudFlow prevents an information leakage as follows:

- A policy is created outlining the fact that workloads run by users Alice and Bob should not be co-resident. The simple policy in Figure 4 can be used as an example. The policy will be described for Alice and Bob and propagated automatically along with both their VMs at time of deployment.
- Sometime in the future Alice is working on client A's data and needs to run her workload.
- Bob uses the same cloud as Alice and it just so happens that the cloud scheduler locates their VMs on the same physical machine *before they both start their respective security-critical workloads*.
- After some time Alice's VM boots up and launches the desired Alice-labeled workload.

- The introspection thread assigned to Alice's VM will immediately raise a flag as soon as it notices the Alice label on the applications. This event is sent back to the PolicyD running on the current physical node. PolicyD looks at the policies that have been specified for the VM that issued the call-back. Although PolicyD finds one policy that specifies Alice, it knows that currently there are no other VMs on the same machine running workloads from Bob hence no action is taken.
- After some more time passes, Bob's VM boots up and launches its Bob-labeled workload. At this point the introspection thread for Bob's VM reports back to PolicyD having seen the desired runtime label 'Bob'.
- PolicyD now works out that an administrator has specified a policy that prohibits the current cloud state (Alice and Bob have co-resident workloads). Alice's VM is frozen to prevent any security breach.
- The action specified by the policy i.e., migration, is then passed back to the PolicyM module by PolicyD along with the id for Alice's VM.
- PolicyM now begins a reconnaissance phase to locate a potential migration target to relocate Alice's VM. In this phase PolicyM queries a list of runtime labels from all potential hosts and as soon as it finds a target host that will not cause a policy conflict with Alice's VM, it returns the host name to the PolicyD that issued the migration request.
- PolicyD, having acquired a conflict-free migration target, issues a migration request to the management module for relocating Alice's VM..
- The OpenStack module issues a migrate action, as it normally would for load balancing, and migrates Alice's VM to the provided migration target, where the VM resumes.

IV. Discussion

Information Flow Invariants. We need to caution that any reactive system cannot be provably secure.

CloudFlow enacts a best-effort approach that aims to minimize the vulnerability window during which an attack can be mounted against a target VM. In practice however, we observe that the window of vulnerability is so small (<5ms, Section V) that it easily defeats any existing and foreseeable side-channel attacks which are usually low-bandwidth and require comparatively ample amount of time to deploy [34, 37, 42] (>hours).

Users & Groups. CloudFlow allows the straightforward definition and enforcement of user and group/role-based policies. Roles can be associated with underlying SELinux labels which can then be used in the policy definitions.

Centralizing Policy Management. Currently CloudFlow is using a database attached to OpenStack as a Policy Store. Whenever a particular VM is being launched, it needs to also attach all policies pertaining to the launch request to be propagated along with its VM launch request. The target VM host thus only needs to keep track of policies that are locally relevant. This significantly reduces the complexity of policy checking required by the local PolicyD instance and reduces latency for migration decisions.

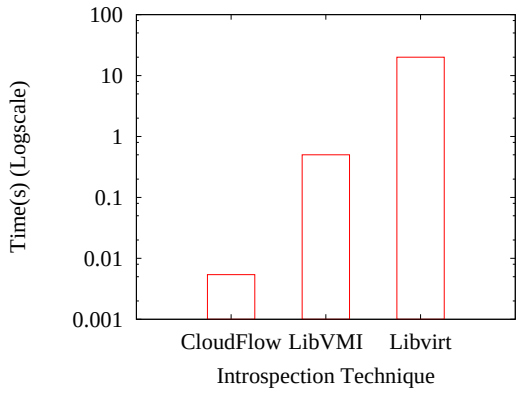


Fig. 7. CloudFlow features a low impact introspection design that allows it to perform significantly faster than existing introspection libraries. The shorter the introspection cycle, the smaller the window of vulnerability during which a VM can be attacked.

The process can be further streamlined by ensuring a default automatic OpenStack-driven policy-to-VM mapping and propagation that can determine applicable policies from the user's session credentials. However, the existence of a set of cloud-wide policy repositories from which users can chose individual subsets of policies of interest may be desirable.

Finally, the policy management system would benefit from a (partial) ordering of runtime labels. This would enable streamlined conflict resolution, e.g., a policy may specify that the "lower" labeled VM should be migrated in the case of triggered inter-VM policy conflicts.

Management & Scheduling Even though CloudFlow is primarily a security tool, the design allows for fine-grained management-oriented usage. The policy engine is oblivious to the semantics of the policies and labels. Policies can be used to describe arbitrary segregations of the in-house cloud in terms of users, workloads and resources. E.g. a CPU intensive workload can be tagged with a corresponding 'CPU-intensive' label. Further VMs with this label are only allowed to migrate to machines that can handle more CPU cycles but may be lacking in storage or other resources. Machines can simply be labeled by tagging custom users or tasks with desired performance numbers for policy usage. Here policy 'conflicts' can be described in terms of performance rather than security. This produces a self-scheduling cloud as VMs automatically end up in places that make sense in terms of real-time resource usage. This kind of scheduling allows VMs to relocate only when needed. VMs can switch between CPU-intensive or I/O-intensive tasks triggering label changes and get scheduled on appropriate nodes transparently.

V. Evaluation

We now explore two different performance aspects of CloudFlow. The first aspect is the ability to minimize the window

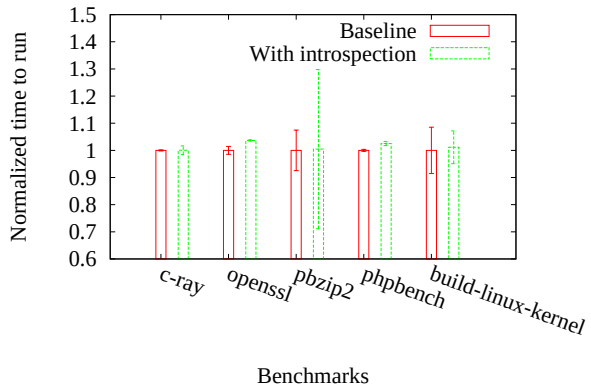


Fig. 8. CloudFlow has a minimal overhead, below 2% slowdown in guest workloads. Performance slowdown caused by CloudFlow remains well within the normal variation shown by typical workloads, represented here by the error bars.

of time during which a particular VM remains vulnerable to side channel attacks, after it has started a security-critical workload. CloudFlow provides a best-effort solution and only a small window of vulnerability exists where the victim can be targeted. The aim of the experiments is to find out exactly how small this window is. The second aspect is the performance slowdown experienced within the guest due to constant introspection. Our introspection technique is polling based and causes a constant but minimal overhead. Finally, VMs may undergo dynamic migration for policy conflict resolution and resultant latency causes the guest to perform slower. We minimize the impact of this overhead by scouting out conflict-free migration targets.

Specs. 2.90 GHz Intel i7-3520M nodes with 4GB memory running 64-bit Ubuntu 12.04.1 LTS. The VMs are running 64-bit Fedora 16 with 2GB of memory. The benchmark suite of choice was the Phoronix Test Suite [7].

Vulnerability Window. Since CloudFlow is reactive in nature, a major aim is to minimize the window of vulnerability during which an attack can be mounted against a target VM. The size of this window is precisely the time it takes from the moment a security critical workload starts to run to the moment an appropriate policy enforcement is executed. Whenever a runtime label matches a policy that needs to be enforced CloudFlow freezes the VM. As CloudFlow performs continuous introspection, the vulnerability window is exactly the wall-clock time it takes to do a complete introspection cycle over the required kernel data structures.

In an initial version of CloudFlow we started using libvirt [29] and libVMI [30] as the libraries of choice for KVM-QEMU introspection. It became quickly apparent that these libraries are not designed for low-latency response and the combination is unusable for meaningful real-time introspection. Instead, we wrote our own custom fast introspection interface for KVM-QEMU which sheds all unnecessary libvmi/libvirt overheads and runs within KVM-QEMU as a thread that can use KVM-QEMU internals directly to read guest memory. This allowed us to *perform orders of magnitude faster that existing techniques*. As shown in Figure 7, CloudFlow only takes around 4.7ms wall clock time (less than 2% of which is CPU time) to perform one complete pass over all the desired kernel structures. This figure was achieved for ¿50 guest processes running – the behavior does not vary significantly with increasing number of processes.

Performance. CloudFlow is broken down into independently functioning components that communicate with each other asynchronously. Out of all the components, the most computationally intensive module is the introspection module. The introspection runs as a thread inside of KVM-QEMU and as a result affects the runtime performance of the monitored guests. Figure 8 shows the runtime performance of the guest operating system with both introspection on and off (lower is better).

The slowdown caused by CloudFlow introspection is upper bound around 1.7% above the baseline performance measured with introspection turned off. CloudFlow performs well within the original error bounds of the baseline performance. All other components run separately and do not cause any performance impact for guest workloads.

Migration Cost. Finally, the guest VM is also subject to dynamic migration in the case of a policy conflict. Whenever this situation arises, the latency of migration causes a performance hit for the guest. For our experimental setup, time taken to do a single migration hovers around the 20 seconds mark – however this figure is likely to vary wildly depending on network load, utilized RAM and volume/disk persistence models (reconnection of volumes to the migrated VM takes time). Although frequent migration of VMs impedes user experience, it is important to note that in most policies, migration likely gets triggered only when a security critical service launches and the VM relocated is mandated by a policy impacting it. The non-cloud alternative of having an entire separate machine dedicated to each individual task is most likely much more expensive.

VI. Conclusion and Future Work

The main contributions of this paper include (i) a new introspection mechanism for the KVM-QEMU [15] hypervisor that allows for fast asynchronous virtual machine introspection at the hypervisor level, orders of magnitude faster than previous approaches, and (ii) a cloud wide information flow control layer for OpenStack [6] that leverages the introspection mechanisms to enforce best effort real-time information flow control.

Ongoing and future work includes (a) new stronger introspection mechanisms resilient to malicious guest OSes and (b) CloudFlow extensions that will allow policies to be written in terms of additional guest state elements, including I/O state, intra-OS events and IPC.

References

[1] Bell-LaPadula model. Online at thttp://en.wikipedia.org/wiki/Bell-LaPadula_model.

[2] Enron Case. Online at http://www.fbi.gov/news/stories/2006/december.

[3] Federal Information Security and Management Act of 2002. Title III of the E-Government Act of 2002. Online at http://www.law.cornell.edu/uscode/text/44/3541.

[4] HIPAA, U.S. department of health and social services. Online at http://www.hhs.gov/ocr/privacy/.

[5] National Association of Insurance Commissioners. 1999. Graham-Leach-Bliley Act. Online at http://www.ftc.gov/privacy/glbact/glbsub1.htm.

[6] Openstack. Online at http://www.openstack.org/.

[7] Phoronix Testing Suite. Online at http://www.phoronix-test-suite.com/.

[8] Sarbanes-Oxley Act of 2002. Online at http://en.wikipedia.org/wiki/Sarbanes-Oxley_Act.

[9] The E-Government Act 04 2002. U.S. Public Law 107-347. Online at http://www.gpo.gov/fdsys/pkg/PLAW-107publ347/pdf/PLAW-107publ347.pdf.

[10] The U.S. Patriot Act. Online at http://www.justice.gov/archive/ll/highlights.htm.

[11] The U.S. Securities and Exchange Commission. 2003. Rule 17a-3&4, 17 CFR Part 240: Electronic Storage of Broker-Dealer Records. Online at http://www.sec.gov/rules/final/34-44992.htm.

[12] The unofficial selinux faq. Online at http://www.crypt.gen.nz/selinux/faq.html.

[13] Sina Bahram, Xuxian Jiang, Zhi Wang, Mike Grace, Jinku Li, Deepa Srinivasan, Junghwan Rhee, and Dongyan Xu. Dksm: Subverting virtual machine introspection for fun and profit. In *Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems*, SRDS '10, pages 82–91, Washington, DC, USA, 2010. IEEE Computer Society.

[14] Fabrizio Baiardi, Diego Cilea, Daniele Sgandurra, and Francesco Ceccarelli. Measuring semantic integrity for remote attestation. In Liqun Chen, ChrisJ. Mitchell, and Andrew Martin, editors, *Trusted Computing*, volume 5471 of *Lecture Notes in Computer Science*, pages 81–100. Springer Berlin Heidelberg, 2009.

[15] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[16] J. Bellessa, E. Kroske, R. Farivar, M. Montanari, K. Larson, and R.H. Campbell. Netodessa: Dynamic policy enforcement in cloud networks. In *Reliable Distributed Systems Workshops (SRDSW), 2011 30th IEEE Symposium on*, pages 57–61, 2011.

[17] Peter M. Chen and Brian D. Noble. When Virtual Is Better Than Real. In *Proceedings of HotOS*, 2001.

[18] P.M. Chen and B.D. Noble. When virtual is better than real [operating system relocation to virtual machines]. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 133–138, 2001.

[19] Wei Chen, Fengqian Gao, and Yuan Lu. Policy based power management in cloud environment with intel intelligent power node manager. In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2012 IEEE 16th International*, pages 66–69, 2012.

[20] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati. Encryption-based policy enforcement for cloud storage. In *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*, pages 42–51, 2010.

[21] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 51–62, New York, NY, USA, 2008. ACM.

[22] Brendan Dolan-Gavitt, Tim Leek, Michael Zhivich, Jonathon Giffin, and Wenke Lee. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 297–312, Washington, DC, USA, 2011. IEEE Computer Society.

[23] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 193–206, New York, NY, USA, 2003. ACM.

[24] Zhongshu Gu, Zhui Deng, Dongyan Xu, and Xuxian Jiang. Process implanting: A new active introspection framework for virtualization. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*, pages 147 –156, oct. 2011.

[25] Mario Heiderich, Marcus Niemietz, Felix Schuster, Thorsten Holz, and Jörg Schwenk. Scriptless attacks: stealing the pie without touching the sill. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 760–771, New York, NY, USA, 2012. ACM.

[26] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 128–138, New York, NY, USA, 2007. ACM.

[27] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 128–138, New York, NY, USA, 2007. ACM.

[28] Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Antfarm: tracking processes in a virtual machine environment. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 1–1, Berkeley, CA, USA, 2006. USENIX Association.

[29] libvirt. The virtualization api.

[30] libvmi. Vmi tools.

[31] K. Nance, M. Bishop, and Brian Hay. Virtual machine introspection: Observation or interference? *Security Privacy, IEEE*, 6(5):32–37, 2008.

[32] Komal Singh Patel and A.K. Sarje. Vm provisioning policies to improve the profit of cloud infrastructure service providers. In *Computing Communication Networking Technologies (ICCCNT), 2012 Third International Conference on*, pages 1–5, 2012.

[33] Nick L. Petroni, Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 13–13, Berkeley, CA, USA, 2004. USENIX Association.

[34] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 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, pages 199–212, New York, NY, USA, 2009. ACM.

[35] Abhinav Srivastava and Jonathon Giffin. Tamper-resistant, application-aware blocking of malicious network connections. In Richard Lippmann, Engin Kirda, and Ari Trachtenberg, editors, *Recent Advances in Intrusion Detection*, volume 5230 of *Lecture Notes in Computer Science*, pages 39–58. Springer Berlin Heidelberg, 2008.

[36] Fei Teng and F. Magoules. Resource pricing and equilibrium allocation policy in cloud computing. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 195–202, 2010.

[37] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on aes, and countermeasures. *J. Cryptol.*, 23(2):37–71, January 2010.

[38] Z. Weinberg, E.Y. Chen, P.R. Jayaraman, and C. Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 147–161, 2011.

[39] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in HiStar. In *Proceedings of the 7th symposium on Operating systems design and implementa-*

tion, OSDI '06, pages 263–278, Berkeley, CA, USA, 2006. USENIX Association.

[40] Ning Zhang, Ming Li, Wenjing Lou, and Y.T. Hou. Mushi: Toward multiple level security cloud with strong hardware level isolation. In *MILITARY COMMUNICATIONS CONFERENCE, 2012 - MILCOM 2012*, pages 1–6, 2012.

[41] Yinqian Zhang, A. Juels, A. Oprea, and M.K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 313 –328, may 2011.

[42] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 305–316, New York, NY, USA, 2012. ACM.