CloudFlow: Cloud-wide policy enforcement using fast VM introspection

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

***Abstract*—Government and commercial enterprises are increasingly considering cloud adoption. Clouds improve overall efficiency by consolidating a number of different clients' software virtual machines onto a smaller set of hardware resources. Unfortunately, this shared hardware also creates inherent side-channel vulnerabilities, which an attacker can use to leak information from a victim VM.**

Side-channel vulnerabilities are especially concerning when different principals are constrained by regulations. A classic example of these regulations are Chinese Wall policies for financial companies, which aim to protect the financial system from illicit manipulation by separating portions of the business with conflicting interests.

Although efficient prevention of side channels is difficult within a single node, there is a unique opportunity within a cloud. This paper proposes a low-overhead approach to cloud-wide information flow policy enforcement: identifying side channels which could potentially be used to violate a security policy through run-time introspection, and reactively migrating virtual machines to eliminate node-level side-channels.

In this paper we describe CloudFlow—an information flow control extension for OpenStack. CloudFlow includes a novel, virtual machine introspection mechanism that is orders of magnitude faster than previous approaches. CloudFlow efficiently and transparently enforces information flow policies cloud-wide, including information leaks through undesirable side-channels. Additionally, CloudFlow has potential uses for cloud management and resource-efficient virtual machine scheduling.

***Keywords*-Cloud computing; Side-Channel Attacks; VM Introspection; Information flow control**

I. Introduction

Cloud computing is an attractive way for companies to flexibly and efficiently share in-house hardware resources across all their users and applications. Unfortunately, clouds also introduce new security concerns; one primary concern is the introduction of side-channels through shared hardware.

When virtual machines (VMs) share hardware, they are vulnerable to malicious side-channel attacks [20, 30, 34, 35, 40], frustrating enforcement of information flow control policies. A major requirement for (in-house) clouds, however, is that workloads from different units may need to be mutually isolated for regulatory and security reasons. For instance, financial companies are required to enforce Chinese wall policies to segregate resources used by different internal departments and provide proof that no information is leaked between departments that could create a conflict of interest. Over 10,000 IT-impacting regulations exist in the US alone, including Sarbanes-Oxley Act [7], Health Insurance Portability and Accountability Act [3], Gramm-Leach-Bliley Act [4], Federal Information Security Management Act [2], Securities and Exchange Commission (SEC) rule 17a-4 [11], the e-Government Act [8], and the Patriot Act [10].

Unfortunately, adoption of cloud technology has outpaced cloud-scale security tools, forcing businesses to deploy inefficient, ad hoc solutions such as manual segregation of internal networks and hosts. Overall, private cloud technology is missing a security platform that can translate high-level policies, such as role-based user and workflow isolation, into runtime enforcement actions throughout the cloud.

We observe that many of these regulations can be modeled as information flow control (IFC) policies, which should be applied at cloud scale. Information flow control (IFC) has been studied extensively at the programming language level [28] OS level [22, 37], and, to some extent, in distributed systems [36] and specialized hardware architectures [33], but has received less attention among VMs in the cloud. Moreover, the side-channel problem introduced by private clouds is a type of covert channel [23], which is difficult for any IFC system to regulate Thus, an important question in adapting any approach to IFC to the cloud is how to limit or prevent node-level side-channels.

Initial work 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 [39], adjusting the timing of access to shared resources [24], or modifying the OS to clean the L1 cache [41].

This paper advances the state of the art by describing CloudFlow, a suite of lightweight OpenStack [5] modules, which automatically mitigate side-channels using VM migration. This work makes the following contributions:

- A new, fast, asynchronous VM introspection mechanism for KVM-QEMU [13], several orders of magnitude faster than previous libraries [25, 26].
- The design of a cloud-wide information flow control layer for OpenStack [5] that leverages the introspection mechanisms to identify risky co-location and mitigate information flow control risks in real time.
- An evaluation of CloudFlow using unmodified guest OSes and applications.

II. Background and Overview

Chinese Wall policies and MLS: In order to explain this work, we use Chinese Wall policies as a motivating

example of important regulatory policies. For instance, a large financial institution may both advise the corporate restructuring of company X and make investment decisions for a mutual fund. The employees advising company X are privy to non-public details that, if passed along to the investment arm, would constitute insider trading. In order for the same company to perform both functions, it must erect a *Chinese Wall* to prevent employees with such a conflict of interest from exchanging information, say in person or over email, that might lead to illicit personal or corporate gain.

Similarly, in intelligence and defense scenarios, information flow control is subject to *Multilevel security* (MLS) policies. MLS policies generally apply to data with different *classification levels*, and prevent things like writing classified data "down" to a public output. One popular example is the well known Bell-LaPadula confidentiality policy model [1].

Previous work on cloud-wide policy enforcement has focused either on MLS for hypervisors [27, 31] or on network isolation and scheduling jobs for better performance [14, 15, 16, 29, 32, 38]. In comparison, CloudFlow provides policy-enforcement across the cloud by leveraging dynamic migration abilities.

Side Channels: In a *side channel attack*, an attacker gains information from a system when the attacker and victim share a resource, such as disks, memory cache, or power conduits. Side channel attacks have been demonstrated which leverage shared caches of physical cloud machines to extract private information such as RSA and AES keys [30, 34, 40]

Approach and Goals: This paper observes that cloud-level side-channel mitigation is a practical alternative to node-level approaches, which sacrifice performance or introduce constraints on the OS or application programmer. CloudFlow uses runtime introspection [18, 21, 26] to identify policy-relevant data usage, and migrates VMs on-demand to avoid the risk of policy violations. 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.

We acknowledge that any reactive system can be subject to *residual side channels*—i.e., the attacker may observe that a policy enforcement action is occurring. In the case of regulatory compliance, residual side channels are not an issue, as the policies themselves and actions of the cloud provider need not be kept secret.

Threat Model & Assumptions: We trust the hypervisor, cloud administrators, and policy designers. All other users are not trusted. Of major concern are insiders who will benefit from cross-VM side channels.

We expect attackers to deploy a malicious VM, through which they will attempt to exploit a known hardware side channel and extract sensitive information from am unsuspecting target VM. We assume that the adversary is not fully in control of the underlying hardware and is unable to bypass hypervisor-enforced isolation of resources such as physical memory or devices.

Our ultimate goal is not to trust guest OSes to participate in policy enforcement; unfortunately, this is not possible with existing introspection techniques, and solving this problem is beyond the scope of the paper. Recent work [12] demonstrates that a malicious guest kernel can deceive hypervisor introspection; thus, the guest must be trusted not to actively subvert the introspection mechanisms. Other approaches to acquiring guest state [17, 19] face similar issues with trust, so the CloudFlow design is independent of how the guest state is acquired, enabling adoption of more robust techniques in the future.

Limitations. No reactive system can be provably secure. CloudFlow provides best-effort minimization of the period during which a side-channel attack can be mounted against a target. In CloudFlow, this window is reduced to $<$5ms (§IV), sufficient to prevent existing and foreseeable side-channel attacks, which are usually low-bandwidth and require a much longer window ($>$hours) [30, 34, 40].

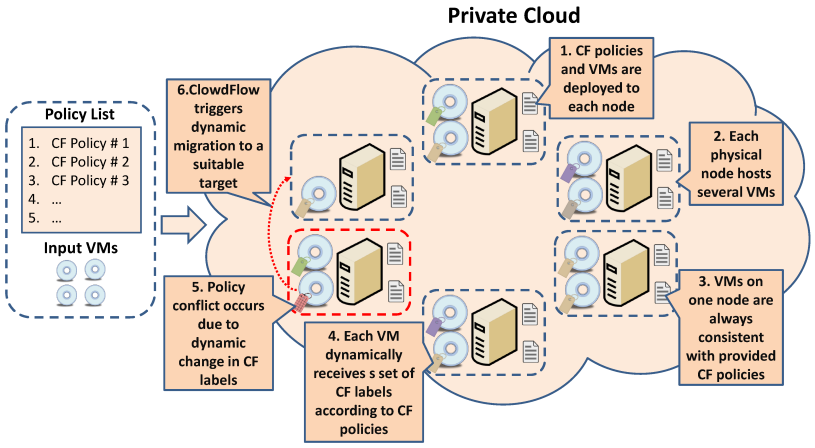
III. Architecture & Implementation

CloudFlow provides a cloud-wide policy enforcement mechanism, integrating with cloud management software to deploy policies to cloud nodes and relocate VMs on demand. The overall CloudFlow design is illustrated in Figure 1a. CloudFlow adds asynchronously connected modules to each node in the cloud and the management interfaces. CloudFlow only requires VM images and policies as input, and policies are propagated to nodes as appropriate during runtime.

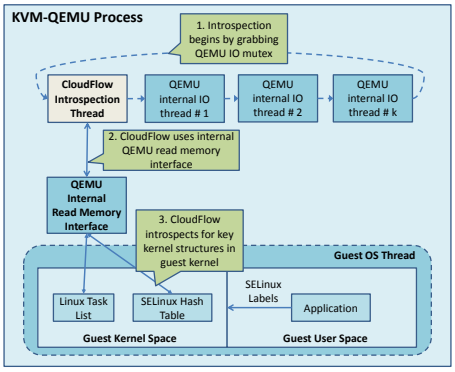
During execution, each VM has a set of application-specific labels associated with its running tasks. If an application accesses policy-sensitive data, or the application exits, the label on the VM will change. When a VM's label changes, this is evaluated against cloud-wide policies in the policy module (§III-C). Label changes that could create a side channel then trigger the migration of one or more VMs to mitigate this risk.

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

The CloudFlow prototype extends the OpenStack [5] framework, version 2012.2. OpenStack offers a variety of services including computing and storage nodes, and works with a wide range of hypervisors. This paper uses KVM-QEMU [13], but works with any hypervisor supported by OpenStack.



(a) Administrators from different departments provide CloudFlow with VM images and desired security policies. CloudFlow (CF) deploys the images and policies to the relevant nodes in the cloud. Introspection is used to dynamically deduce CF labels for VMs. Labels and policies are used by the management module to migrate VMs to mitigate side-channel risks.



(b) CloudFlow deploys a fast introspection thread that uses KVM-QEMU internals to efficiently infer information about tasks running inside the guest OS.

Figure 1: CloudFlow architecture and introspection design

A. *Introspection Module*

CloudFlow deploys VM introspection to extract runtime labels from key guest kernel data structures. In the initial prototypes, we found that the high-latency of existing introspection libraries created unacceptably large windows for attack (detailed in §IV). Upon further investigation, the generality of these libraries comes at a cost of several, expensive layers of indirection. Thus, a key contribution of this work is a library that more efficiently hooks into KVM-QEMU internals, significantly reducing VMI overheads.

CloudFlow adds a specialized introspection thread to each hypervisor. KVM-QEMU schedules this thread in the same class as internal hardware I/O threads that implement hardware I/O virtualization (Figure 1b). The introspection thread also uses KVM-QEMU-internal locking and coordinates with internal events, permitting seamless inter-operation with other internal I/O threads. From this vantage point, our introspection module has fast access to guest memory.

The introspection thread periodically monitors two key guest kernel data structures for changes: the task list and the SELinux hash table containing policy information for all running tasks. The policy module gives the introspection thread a set of policy-relevant labels, which the introspection thread continuously searches for in the guests. If the introspection thread notices a relevant label change inside the guest, this information is sent to the policy module within a few milliseconds.

B. *Policy Language Model*

The CloudFlow policy language is a simple, XML-based grammar, illustrated in Figure 2, although any arbitrary Turing-complete language would suffice. Each policy list

```
<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>
```

Figure 2: Subset of CloudFlow Policy Language

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.

Our prototype uses SELinux labels [9] in order to leverage the rich ecosystem of existing policies. We hasten to note that our policies are much simpler to write than SELinux policies. Our design is not tightly coupled with SELinux, and could be extended to use other labeling schemes, OSes, and security frameworks.

Intuitively, the policy symbols outline the action that needs to be taken if a VM contains the label specified by the policy. 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>.

For example, suppose users Alice and Bob are supposed to be separated by a Chinese Wall policy—the corresponding policy for Bob is illustrated in Figure 3. In this example, processes run by Alice are marked as “Alice” by SELinux in each guest, which would then be observed by introspection, leading to a VM migration as needed.

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

Figure 3: Example policy

C. Management Module

The management module is the administrative interface to CloudFlow, integrated with the OpenStack platform. The administrator inputs a VM image along with a list of security policies. In addition to the usual management tasks, such as deploying and tracking VMs, the CloudFlow management module stores and propagates policies to nodes, and handles policy-related call-backs for events such as VM migration.

D. Policy Module (Daemon, Master)

The CloudFlow management module propagates policies to the policy enforcement infrastructure. CloudFlow runs a cloud-wide policy master module (PolicyM) and a policy daemon on each node (PolicyD). PolicyM coordinates all running instances of PolicyD, as well as tracking which physical hosts are running VMs with a given label. At each node, PolicyD maintains policy lists and compares these policies with the output of the introspection module.

When PolicyM passes a new policy to PolicyD, PolicyD updates the introspection module's list of labels to search for. PolicyD also notifies PolicyM when a new VM is started or stopped on its node. If the introspection module finds a label of interest, the introspection module notifies PolicyD to take action, as specified in the policy. Actions include "halt", which calls the management module, or "migration", which calls PolicyM to identify a suitable destination.

When PolicyM needs to migrate a host, it selects a node that, at least in the immediate future, will not risk further side channels. PolicyM first checks its cache of cloud-level state, looking for suitable destinations. Before migrating a VM to a node, PolicyM queries the node's PolicyD to ensure that the set of labels is still suitable. PolicyM then returns the selected node to PolicyD, which calls the management module to request VM migration.

E. Illustrative Scenario

Figure 4 illustrates a law firm with representing clients A and B in a suit against each other, handled by employees Alice and Bob, respectively. State and federal laws prevent Alice and Bob from sharing any information to ensure both clients are fairly represented. 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, such as the one in Figure 3.
- Alice starts a VM in the cloud.
- Bob uses the same cloud as Alice, and the cloud scheduler places Bob's VMs on the same physical machine *before they start their security-critical workloads*.
- After some time Alice's VM launches an Alice-labeled workload.

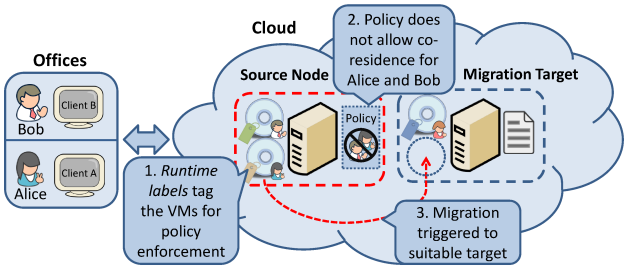


Figure 4: CloudFlow performs dynamic policy-triggered VM relocation based on input policies. An example policy is one that disallows two users (Alice and Bob) from being co-resident on the same physical node based on *runtime labels*.

- The introspection thread will raise a flag as soon as it notices Alice's label. This event is sent to the node's PolicyD, which checks the current policies. Although PolicyD finds a policy that involves Alice, it knows that currently there are no other VMs on the same machine running workloads from Bob, hence no action is taken.
- Bob's VM then launches a Bob-labeled workload. The introspection thread reports the presence of label 'Bob' to PolicyD.
- PolicyD identifies a policy violation: One of the two conflicting VMs is picked at random (in this case Alice's VM) and frozen to prevent any security breach.
- Because the policy specifies migration, PolicyD asks PolicyM for a suitable target.
- PolicyM identifies a potential target for Alice's VM, which does not currently run any of Bob's VMs. PolicyM returns a node name to PolicyD.
- PolicyD issues a migration request to the management module for Alice's VM.
- The OpenStack module migrates Alice's VM, using the same infrastructure designed for load balancing, where the VM resumes.

IV. Evaluation

We evaluate CloudFlow using a set of 5 nodes with a 2.90 GHz Intel i7-3520M CPU and 4GB memory running 64-bit Ubuntu 12.04.1. The VMs are running 64-bit Fedora 16 with 2GB of memory. We exercise the system with the Phoronix Test Suite [6].

Vulnerability Window. An essential goal of CloudFlow is minimizing 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 until the moment an appropriate policy action 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.

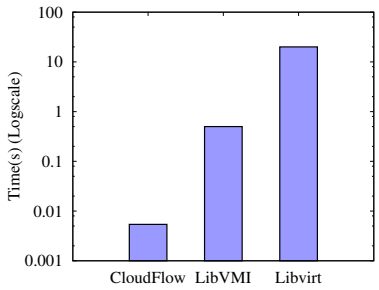


Figure 5: CloudFlow's low-latency introspection design scans kernel data structures significantly faster than existing VMI libraries.

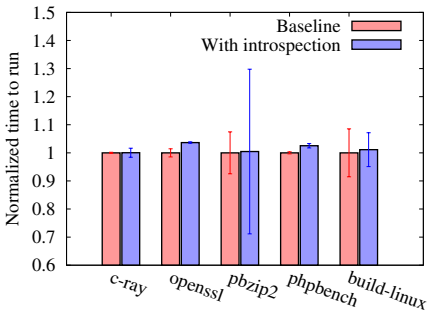


Figure 6: CloudFlow introduces under 4% overhead into guest workloads.

We initially adopted libvirt [25] and then libVMI [26] for KVM-QEMU introspection. As illustrated in Figure 5, these libraries are not designed for low-latency data structure searches, rendering them unsuitable for our purposes. Thus, we wrote a faster introspection interface for KVM-QEMU which sheds all unnecessary indirection, running within KVM-QEMU. Figure 5 measures the time to read all running tasks and SELinux labels with >50 guest processes running. CloudFlow's introspection module requires only 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 is orders of magnitude faster than the alternative libraries. This execution time does not vary significantly with increasing number of processes.

Performance. CloudFlow's most CPU-intensive component is the introspection module, and thus the most likely to affect guest performance. Figure 6 shows the runtime performance of the our benchmarks, with both introspection on and off (lower is better), averaged over 3 runs.

The maximum observed slowdown caused by CloudFlow introspection is less than 4% of 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. In our testbed, a migration adds a 20 second delay, although this number will vary widely depending on the cloud hardware and load. Although frequent migration of VMs harms the user experience, we note that, in most policies, migration only occurs when necessary to uphold security policies. We expect that our design provides a more efficient alternative to dedicated hardware.

V. Conclusion and Future Work

This paper describes CloudFlow, a system which leverages a novel, low-latency introspection mechanism to enforce best-effort, cloud-wide information flow policies in the OpenStack framework. Future work includes introspection mechanisms that are resilient to a malicious guest OS, as well as CloudFlow extensions to permit policies on additional guest abstractions, such as I/O and IPC.

Acknowledgements

We thank Bhushan Jain for help with the CloudFlow implementation. We thank the anonymous reviewers for their insightful comments on earlier versions of this paper. This research was supported in part by NSF grants CNS-1149229, NSF CNS-1161541, NSF CNS-1228839, NSF CNS-1318572, NSF CNS-1223239, NSF CCF-0937833, by the US ARMY award W911NF-13-1-0142, and by gifts from Northrop Grumman Corporation, Parc/Xerox, Microsoft Research, CA, and the Office of the Vice President for Research at Stony Brook University.

References

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

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

[3] HIPAA, U.S. department of health and social services. http://www.hhs.gov/ocr/privacy/. §I

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

[5] Openstack. http://www.openstack.org/. §I, §III

[6] Phoronix Testing Suite. http://www.phoronix-test-suite.com/. §IV

[7] Sarbanes-Oxley Act of 2002. http://en.wikipedia.org/wiki/Sarbanes-Oxley_Act. §I

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

[9] The UnOfficial SELinux FAQ. http://www.crypt.gen.nz/selinux/faq.html. §III-B

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

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

[12] 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 *SRDS*, pages 82–91, 2010. §II

[13] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *ATEC*, pages 41–41, 2005. §I, §III

[14] J. Bellessa, E. Kroske, R. Farivar, M. Montanari, K. Larson, and R.H. Campbell. Netodessa: Dynamic policy enforcement in cloud networks. In *SRDSW*, pages 57–61, 2011. §II

[15] Wei Chen, Fengqian Gao, and Yuan Lu. Policy based power management in cloud environment with intel intelligent power node manager. In *EDOCW*, pages 66–69, 2012. §II

[16] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati. Encryption-based policy enforcement for cloud storage. In *ICDCSW*, pages 42–51, 2010. §II

[17] Yangchun Fu and Zhiqiang Lin. Space traveling across VM: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection. In *Oakland*, pages 586–600, 2012. §II

[18] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP*, pages 193–206, 2003. §II

[19] Zhongshu Gu, Zhui Deng, Dongyan Xu, and Xuxian Jiang. Process implanting: A new active introspection framework for virtualization. In *SRDS*, pages 147–156, 2011. §II

[20] Mario Heiderich, Marcus Niemietz, Felix Schuster, Thorsten Holz, and Jörg Schwenk. Scriptless attacks: stealing the pie without touching the sill. In *CCS*, pages 760–771, 2012. §I

[21] Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Antfarm: Tracking processes in a virtual machine environment. In *ATEC*, pages 1–1, 2006. §II

[22] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. Information flow control for standard OS abstractions. In *SOSP*, 2007. §I

[23] B. W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10):613–615, 1973. §I

[24] Peng Li, Debin Gao, and Michael K. Reiter. Mitigating access-driven timing channels in clouds using StopWatch. In *DSN*, pages 1–12, 2013. §I

[25] libvirt. The virtualization api. http://libvirt.org/. §I, §IV

[26] libvmi. Vmi tools. https://code.google.com/p/vmitools/. §I, §II, §IV

[27] Robert Meushaw and Donald Simard. Nettop: Commercial technology in high assurance applications. *Tech Trend Notes: Preview of Tomorrows Information Technologies*, 2000. §II

[28] A. C. Myers and B. Liskov. A decentralized model for information flow control. In *SOSP*, pages 129–142, October 1997. §I

[29] Komal Singh Patel and A.K. Sarje. VM provisioning policies to improve the profit of cloud infrastructure service providers. In *ICCCNT*, pages 1–5, 2012. §II

[30] 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 *CCS*, pages 199–212, 2009. §I, §II, §II

[31] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J.L. Griffin, and L. van Doorn. Building a MAC-based security architecture for the Xen open-source hypervisor. In *ACSAC*, pages 276–285, 2005. §II

[32] Fei Teng and F. Magoules. Resource pricing and equilibrium allocation policy in cloud computing. In *CIT*, pages 195–202, 2010. §II

[33] Mohit Tiwari, Hassan M.G. Wassel, Bita Mazloom, Shashidhar Mysore, Frederic T. Chong, and Timothy Sherwood. Complete information flow tracking from the gates up. In *ASPLOS*, pages 109–120, 2009. §I

[34] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on AES, and countermeasures. *J. Cryptol.*, pages 37–71, 2010. §I, §II, §II

[35] 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 *Oakland*, pages 147–161, 2011. §I

[36] N. Zeldovich, S. Boyd-Wickizer, and D. Mazières. Securing distributed systems with information flow control. In *NSDI*, 2008. §I

[37] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in HiStar. In *OSDI*, pages 263–278, 2006. §I

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

[39] Yinqian Zhang, A. Juels, A. Oprea, and M.K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Oakland*, pages 313–328, 2011. §I

[40] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-VM side channels and their use to extract private keys. In *CCS*, pages 305–316, 2012. §I, §II, §II

[41] Yinqian Zhang and Michael K. Reiter. Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *CCS*, pages 827–838, 2013. §I