Do You Know Where Your Data Are?
Secure Data Capsules for Deployable Data Protection

Petros Maniatis[†], Devdatta Akhawe[*], Kevin Fall[†], Elaine Shi[*‡], Stephen McCamant[*], Dawn Song[*]
[‡]*PARC,* [*]*UC Berkeley,* [†]*Intel Labs Berkeley*

1 Data Protection But Not For All

Do you know where your data are? Who can see them? Who can modify them without a trace? Who can aggregate, summarize, and embed them for purposes other than yours? We don't, and we suspect neither do you. The problem is that we do not have a widely-available mechanism to answer these questions, and yet, paradoxically, all evidence shows that it should have been solved long ago. The problem is critical; incidents involving sensitive data leakage, unauthorized access, and integrity violations (accidental or not) are a daily occurrence [1]. It is well known, as evidenced by the volume of relevant government regulation and pontification from privacy advocates. It is interesting, since it has inspired much research into data confidentiality, integrity, and authorization. Yet publicizing it, regulating it, and talking about it have not led to solving the problem effectively for the vast majority of users. Why?

We believe the root of this paradox lies in the disconnect between research, policy, and industry objectives, and the needs of the real world. Research has largely focused on elegance and intellectual exploration, while industry has built expedient solutions for media content protection and enterprise rights management, guided by projected revenue. In either case, the problem is solved for some users, under narrow scenarios, but neither research results nor rights management systems have resulted in broad applicability and deployment. In this paper, we examine the problem of protecting data for *all* users, and not just for some. We use broad applicability as the driving goal, and explore the challenges and promises of reaching towards that goal with efficiency and high assurance.

To be broadly applicable, a data protection mechanism must support, first and foremost, *backward compatibility*. Any mechanism must seamlessly integrate with unmodified applications and data formats. Even in the face of vulnerabilities, pragmatic concerns make it tough to argue for the wholesale migration of systems from existing infrastructure and legacy programs to something completely new. Research proposals that require completely new operating systems [11, 32], languages [21], or both [25], do not apply to legacy systems and therefore contribute only indirectly to solving our problem.

Beyond the ability to support existing applications, a data protection mechanism should be *flexible*. Solutions based on a centrally vetted, closed set of applications are incompatible with today's diverse, dynamic systems. A prime example are commercially available enterprise rights management (ERM) solutions. Although technical details and scrutiny are scarce, ERM products appear to be based on a common proprietary application framework [3, 20] or on application-specific modifications [12]. Consumerization, which fosters individual choice, leads many users to find the "walled gardens" of ERM systems unacceptable. On the other hand, the limited set of supported applications inhibits interoperability among enterprises—a cost organizations often refuse to bear. When an organization does decide to move to a different application platform, it may risk losing continuity within its own documents, at a potentially astronomical cost.

As a motivating example, consider protecting the privacy and integrity of a patient's—we will call him Owen—personal health records. In the USA, the privacy of these records is a requirement laid down by statute, with significant penalties for violations [2]. Their integrity is also critical: unauthorized modifications could be fatal—consider the inadvertent removal of Owen's alergies from his record. Current usage exhibits significant complexity. Owen's record may be handled by all the physicians who treat or advise him, all of his insurers over time and employment changes, and all the hospitals and clinics where he is seen. This usually large set of people and organizations view and modify Owen's record on varied platforms (clinic PCs, mobile tablets, web forms), managed by IT staff of varying skill. It is unreasonable to expect such a diverse conglomeration of users and systems to change *en masse* into a new, common OS and a small set of blessed applications. At the same time, restrictive solutions like ERMs are too specific, and do not work seamlessly across hospitals with distinct infrastructures (e.g., when Owen is treated while on vacation abroad). Although representative in criticality and complexity, this example is by no means unique. From social networking to advertising systems, from financial transactions to email records, examples abound, each with varying privacy and integrity requirements.

2 The Secure Data Capsules Vision

To bring deployable data protection to a broader base, we envision an evolutionary conceptual architecture based on secure data encapsulation. A sensitive data object is kept in a *secure data capsule*, cryptographically bundled with its *data-use controls* (DUCs) policy and provenance.

This paper appears in HotOS 2011.

Owen's health record, in our example, would be contained within a capsule with a DUC specifying "allow any doctor to view and Charlie to modify," among other restrictions, and provenance with entries like "Charlie appended a prescription on May 11th." DUCs may cover confidentiality, integrity, non-repudiation, logging, access control, etc. Maintaining integrity, with appropriate evidence, allows a user of a capsule to place trust in its veracity. Maintaining confidentiality affords the data owner or author an assurance of privacy.

It is a fundamental principle of our vision that, unless explicitly prohibited, any application may perform operations on a capsule, as long as those operations are assuredly compliant with policy and appropriately recorded in provenance. Untrusted applications operate on capsules confined within a secure execution environment, which tracks sensitive information flows and enforces policy upon externalization from the application. In our example, anyone would be able to use any application on Owen's record. However, when trying to write to a file or send over the network, only modifications by Charlie would be allowed (and recorded in provenance); the secure execution environment would reject all modifications by others on Owen's health record. What is more, this restriction would apply not only to the original record file, but also to any data object derived from Owen's record—say if Charlie copied an MRI image from Owen's record to a new image file; if allowed, the derivative data object would also be encapsulated with the same data-use controls as Owen's record.

Our threat model consists of untrusted OSes and applications operating on capsules. We consider an adversarial Charlie, human or organization, but limit his sophistication. We wish to prevent him from accidentally, intentionally, or unknowingly violating policy or evading provenance. We are not concerned here with fixing the "analog hole" (where Charlie takes a photograph of protected capsule contents from the screen and types them in anew), or closing all possible covert channels that might leak information. Those are important considerations, but they require social as well as technical assistance.

Although this conceptual architecture reaches the practicality goal by definition, the challenge of achieving that goal with efficiency (no more than 2x slowdown of applications) and high assurance (a Trusted Computing Base orders of magnitude smaller than commodity OSes) entails a number of research questions, and opportunities. In what follows, we define practical data protection and substantiate gaps in current theory and practice.

3 Why Is It Unsolved?

We now briefly address why prior research efforts have not achieved practicality, efficiency and high assurance.

Practicality is not satisfied by systems that have no legacy support. Systems such as Asbestos and HiStar [11, 32] aim for a small TCB but offer new system interfaces and require applications to be rewritten. A hybrid system such as Flume [17], which provides standard OS abstractions for an Asbestos-like kernel, is more practical, but suffers from a large TCB—including the entire Linux kernel—reducing assurance. Similarly, approaches that require new languages [21] or recompilation from source with annotations [31] imply universal application porting, an uncommon and error-prone process. Finally, commercial ERM systems that require per-application modifications or a single proprietary framework fail on interoperability and openness grounds, and are proprietary so offer few opportunities to assess assurance.

Our high-assurance goal is most helped by approaches that minimize the TCB, most notably by removing from it the OS and application, primarily via virtualization. TrustVisor [19] has a very small TCB, but still requires trusted pieces of application logic to register themselves via hypercalls. Proxos [27] proxies sensitive system calls to a stub OS, but requires non-trivial application porting. Overshadow [7] extends a VMM to present an encrypted and integrity-protected version of an application's memory to the OS without application modifications, but requires application trust. While closer, these approaches still do not achieve our practicality goal.

Bringing together practicality and a small TCB, approaches that use dynamic information-flow tracking to precisely protect sensitive data and their derivatives, while trusting only a small information-flow tracker, seem to be the closest to our goals, in principle. In practice, most information-flow tracking solutions that require no application modifications are inefficient. Overhead reductions have been achieved by systems like Neon [33], which uses on-demand emulation of tainted instructions to track fine-grained information flow for multiple principals' sensitive data across applications and kernel. PIFT [13] further reduces overhead by tracking information flow asynchronously. Unfortunately, these tools dynamically analyze the whole image of a virtual machine (kernel and user space), which leads to significant loss in precision and safety due to kernel complexity [26]. What is more, to improve efficiency, they must trust applications somewhat (e.g., not to leak information via control flows).

4 A Conceptual Architecture

Our constraints of practicality, efficiency and high assurance induce a number of properties on *any* solution.

First, dealing with arbitrary legacy binaries entails reference monitor-like isolation mechanisms; redesign or recompilation are inapplicable. Secondly, efficient support for applications handling sensitive and insensitive data in complex variegated flows necessitates information flow tracking. The inclusion of OSes and commodity appli-

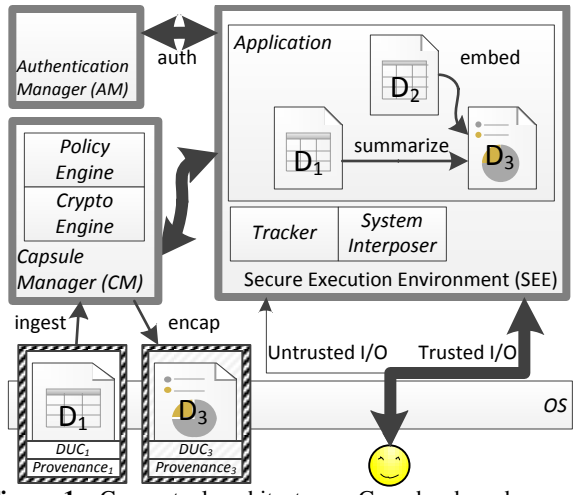

Figure 1: Conceptual architecture. Gray borders demarcate trusted components. Diagonal borders mark capsule boundaries enforced cryptographically. Thick arrows are secure channels.

cations into the TCB is incompatible with our high assurance goal. In their absence, validation of components like the reference monitor and flow tracker will require mechanisms like remote attestation and authentication, preferably with a hardware root of trust. Fourth, cryptographic protection of confidentiality and integrity of stored data is necessary to exclude systems that only traffic data (e.g., networks, storage) from the TCB. Finally, since cryptography among distinct parties will be employed, a trusted, isolated component that deals with credentials, authentication, and key-material storage is required, which we call the *Authentication Manager* (AM).

The above constraints naturally lead to the conceptual architecture in Figure 1. The data unit in this architecture is the secure data capsule, a container comprising a data object, its DUC, and its history. The capsule is cryptographically protected for confidentiality and integrity.

Systems manipulate capsules via three trusted logical components. Besides the AM mentioned above, the *Capsule Manager* (CM) deals with making policy decisions and accordingly releasing clear-text data objects to the application and re-packaging outgoing data. The Secure Execution Environment (SEE) is the reference monitor and information flow tracker that performs a controlled execution of untrusted applications. It interposes on system calls, and interacts with the user via a trusted I/O channel. Outside the TCB are the OS, user (Charlie), and application. Recall that our threat model treats Charlie as a sink, as we place the "analog hole" problem out of scope.

Continuing with our example, if Charlie (Owen's doctor) wants to access Owen's health records, he launches the application of his choice in a SEE. The SEE authenticates Charile (via the AM) and starts execution. It traps any attempt by the application to read a capsule and transparently queries the CM for authorization. The CM evaluates the capsule's DUC against Charlie's credentials, and if the read is permitted, decapsulates the object and provides it to the SEE over a secure channel. Execution can then proceed as normal inside the SEE. A similar protocol is followed for other actions (e.g., writes).

Ensuring confidentiality requires the SEE to track flow of sensitive information to any externalization point (e.g., disk writes, network output). The SEE must send any sensitive data (or their derivatives) to the CM before externalization. The CM subjects them to the originating capsule's policy, and can even default to a restrictive policy. For example, a sensible default would be for the CM to compute a new policy from the DUCs of all sensitive inputs to the application, update the history based on the tracked operations and return a new capsule to the SEE to complete the externalization. In our example, that would mean composing both Owen's and Pam's (another patient) policies on a report Charlie puts together from their records.

The broad architecture outlined above achieves practicality. The TCB can be implemented in commodity OSes and support arbitrary applications. It leaves, however, a number of design and implementation challenges open, particularly given our goals of efficiency and assurability. We explore challenges, and articulate promising research directions in the next section.

5 Challenges and Agenda

We present the major challenges and questions elicited by our architecture, and directions we plan to explore.

5.1 Efficient Information Flow Tracking

The most significant challenge implied by the conceptual architecture is how to perform the tracking task of the SEE at a fine granularity yet with high performance. Fine-grained information flow tracking has a poor performance track record, but we anticipate opportunities for significant overhead reductions along several fronts. Achieving them without compromising correctness and assurance is a challenging research direction.

Restriction. Limiting information flow analysis to a part of the running system can improve performance by reducing taint-spread and tracking overhead. For example, information flow tracking can be restricted to only user space if we can ensure that the kernel can never read sensitive data (e.g., via a security visor [7]). Tracking can be restricted further within the application, if one can isolate application pieces that touch sensitive data (where flow tracking is needed) from those that do not (where tracking is unnecessary). As with all isolation, this compartmentalization might increase communication overheads across application compartments. *Question 1: Is it possible to automatically and efficiently identify pieces of an application binary for which tracking is unnecessary? Question 2: Is it possible to, dynamically*

or statically, optimize the overhead of this partitioning?

Granularity. A related opportunity comes with granularity adjustment. Several proposals have shown the (relative) performance benefits of dynamically switching the tracking granularity from page-level to byte-level [13, 14, 33], or even to module-level. As above, it would be interesting to partition an application into components that benefit from tracking at different granularities. As tracking granularity only affects performance, not safety, this can even be a dynamic, adaptive process over the course of execution. Furthermore, the choice of granularity might be decided by the intrinsic structure of an application's information flow graph. Applications consisting of components shared across principals, leading to a "high-mixing" information flow graph, might necessitate finer granularity. *Question 3: Can an application binary be automatically partitioned into components tracked at different granularities so as to balance the cost of "over-protection" with the overhead of precise tracking? Question 4: Can the information flow graph of an application be characterized in a concise manner sufficient for modeling the cost of tracking granularity?*

Asynchrony. Benefits of asynchronously tracking information flow have been demonstrated for both hardware [6, 16] and software [13] systems. The idea is to buffer a trace of the instructions executed as well as their inputs and outputs, and perform the analysis of that trace at a later time, before a policy enforcement decision must be taken (e.g., at system calls). The performance benefit comes from reduction in contention (e.g., of the cache) between the application and the tracking instructions. A natural question is whether this duality between tracing and tracking can be stretched, e.g., deferring the tracking computation on a trace until a capsule is about to be read, possibly at a different machine. This may be useful for expensive analyses [4, 18]. *Question 5: Can the computation-storage trade-off between eager and lazy information flow tracking be harnessed for good?*

Hardware. Direct hardware support and optimizations have been shown, for some research architectures, to improve overall performance significantly [6,8,16,28]. Pragmatic questions remain however. *Question 6: Is hardware acceleration for information flow tracking relevant to modern hardware architectures?* Three factors appear encouraging. First, off-loading flow tracking support to a co-processor [16] may avoid main CPU modifications. Second, some popular architectures already contain considerable structures for tracking data and control dependencies [15, 22]. Use of these may limit the degree of hardware modifications required for implementing flow tracking. Third, hybrid designs of popular architectures, such as Transmeta's Crusoe, use binary emulation in firmware [9], a natural place "close to the metal" to provide information flow tracking with amortized overheads.

5.2 Effective Isolation

It is essential that trusted components maintain their integrity and confidentiality despite user or OS tampering. Prior isolation mechanisms [5, 24, 27, 29] mostly treat an entire virtual machine as the unit of isolation, which is coarser than our problem definition requires. In principle, per-application VMs might be an acceptable compromise [23, 24], but would require the SEE to "touch" OS instructions, which appears to be expensive and unnecessary. There is no need for an OS to know the contents of an application's memory pages. It appears that cryptography-based memory-masking techniques—e.g., Overshadow's does not even require application modifications—are the most promising, but have only been shown for standard VMMs with significant code size. *Question 7: Is fine-granularity, efficient, small-TCB isolation possible?* We see two opportunities. First, we require significantly less functionality than a general-purpose VMM: only a single guest, which means no need for optimized VM switches, inter-VM communication, memory deduplication, etc. Removing application trust from small security visors [19] might prove sufficient. Second, extending existing virtualization support in hardware to deal with threads or thread pieces, rather than VMs is plausible. Isolation is usually implemented with *hardware range registers* guarding page ranges, which should be extensible to finer-granularity ring-3 functionality. Some public announcements from microprocessor manufacturers seem to foreshadow efficient, fine-granularity, hardware-enforced isolation [30].

5.3 Protecting The Sum of Many (Protected) Parts

Many important computations, especially in web services, analyze myriad inputs, possibly from distinct users' capsules. While in theory the output of such massive computations over large data sets reveals little of the input information, enforcing data owners' DUCs effectively is challenging and insufficiently defined.

A common proposed solution is differential privacy [10], in which, conceptually, a query engine takes a data capsule and adds appropriate noise to the output of a computation on those contents. Since data may come from different owners, with possibly different privacy requirements, one challenge is how to manage and optimize for potentially heterogenous privacy budgets. *Question 8: How can we manage and reconcile different users' privacy requirements in a common computation?* One promising idea is to manage or auction the privacy budget as a form of resource. Another idea is to exploit the fact that many real-world databases are constantly evolving, and queries tend to focus on the latest trends. In these settings, we can periodically time out stale data, and always use the latest data in queries. This potentially bounds the number of queries on each data item, and thereby avoids

depletion of the privacy budget.

Another question is how the differential privacy engine can decide how to sanitize the data based on the history of the data object. A popular approach for noise addition is to add noise proportional to the sensitivity of the statistic. *Question 9: Can we automatically determine the sensitivity of a legacy program and transform it into its differentially private counterpart?*

5.4 What Is the Policy About?

The semantic richness of policy impacts the design of any policy language, tracker, and enforcer. For example, it is moot for Owen to control properties of his data (e.g., "do not shrink" for an image) unless the CM (the enforcer) and the SEE (the tracker) understand shrinking. In the simplest case—which requires the least application-specific understanding within the TCB—policy is in terms of information flow properties, such as **read**, **update**, and **forward**, **excerpt** (propagate the object partly), and **paste** (embed the object in another). When application-specific properties must be controlled, e.g., image shrinking, the trusted components must be extended with new property semantics, even after deployment (as per the practicality goal). *Question 10: Is extensible policy semantics feasible without TCB inflation?*

One promising insight is that a property checker can be simpler than a full-blown application; a proof-carrying implementation of a property checker may be feasible. Another compromise is to make the safety of application-specific properties optional; while information-flow policy is guaranteed, an owner with application-specific demands might only obtain guarantees conditioned on the correctness of associated checkers. However, making this distinction evident to the user may be challenging. *Question 11: Can an extensible, variable-trust policy framework be intuitive to unsophisticated users?*

An easier problem may be tracker extensibility. In the shrinking example, a particular application can be piece-meal instrumented to provide (untrusted) hints about shrinking, subsequently checked by the enforcer. *Question 12: Is it feasible to instrument binaries for application-specific tracking, without source code?*

5.5 The Road Ahead

Our discussion here has focused on challenges with major, open research questions. But many other engineering challenges must be addressed to get from theoretical practicality to practical practicality. First, OSes deal in memory pages and buffers, not in objects; identifying object boundaries will require careful design, as will dealing with complex system call semantics and other OS communication abstractions. Second, cryptographic overhead could be significant, fragmented across many small operations, and spread over large numbers of keys. Clever caching and transcryption might help reduce the cost. Third, the precise cryptographic design of our approach—e.g., what keys to use to encrypt capsules—is itself intriguing; one may seal capsules only to the trusted components without requiring a PKI for principals, but naming authorities might still be necessary (e.g., defer to a hospital to define Owen's assigned physicians).

On a more theoretical level, solving our problem in the face of covert channels (between a colluding application and the OS, for example) is probably beyond feasibility. We hope, however, to study mitigating mechanisms at the SEE to reduce the capacity of known covert channels (e.g., timing, memory contention, etc.).

Our agenda is ambitious. We are optimistic that progress along any of the open questions will bring us closer to practical data protection for all.

References

[1] DataLoss DB: Open Security Foundation. http://datalossdb.org.

[2] 104th Congress, United States of America. Public Law 104-191: Health Insurance Portability and Accountability Act (HIPAA), 1996.

[3] Adobe Systems Inc. LiveCycle Rights Management. http://www.adobe.com/products/livecycle/rightsmanagement/, Jan. 2011.

[4] M. Backes, B. Köpf, and A. Rybalchenko. Automatic Discovery and Quantification of Information Leaks. In *IEEE Security and Privacy*, 2009.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP*, 2003.

[6] S. Chen, M. Kozuch, P. B. Gibbons, M. Ryan, T. Strigkos, T. C. Mowry, O. Ruwase, E. Vlachos, B. Falsafi, and V. Ramachandran. Flexible Hardware Acceleration for Instruction-Grain Lifeguards. *IEEE Micro Magazine*, 29, 2009.

[7] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. K. Ports. Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems. In *ASPLOS*, 2008.

[8] M. Dalton, H. Kannan, and C. Kozyrakis. Raksha: A Flexible Information Flow Architecture for Software Security. In *ISCA*, 2007.

[9] J. C. Dehnert, B. K. Grant, J. P. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson. The Transmeta Code Morphing Software: Using Speculation, Recovery, and Adaptive Retranslation to Address Real-Life Challenges. In *CGO*, 2003.

[10] C. Dwork. Differential Privacy. In *ICALP*, 2006.

[11] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, F. Kaashoek, and R. Morris. Labels and Event Processes in the Asbestos Operating System. In *SOSP*, 2005.

[12] EMC Corp. Documentum Informaton Rights Management. http://www.emc.com/products/detail/software/information-rights-management.htm, Jan. 2011.

[13] A. Ermolinskiy, S. Katti, S. Shenker, L. Fowler, and M. McCauley. Towards Practical Taint Tracking. Technical Report UCB/EECS-2010-92, UC Berkeley, June 2010.

[14] A. Ho, M. Fetterman, C. Clark, A. Warfield, and S. Hand. Practical Taint-Based Protection using Demand Emulation. In *EuroSys*, 2006.

[15] Intel Corporation. Intel Itanium Processor 9000 Sequence. http://www.intel.com/products/processor/itanium/index.htm.

[16] H. Kannan, M. Dalton, and C. Kozyrakis. Decoupling Dynamic Information Flow Tracking with a Dedicated Coprocessor. In *DSN*, 2009.

[17] 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.

[18] S. McCamant and M. D. Ernst. Quantitative Information Flow as Network Flow Capacity. In *PLDI*, 2008.

[19] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *IEEE Security and Privacy*, 2010.

[20] Microsoft Corp. Windows Rights Management Services. http://www.microsoft.com/windowsserver2008/en/us/ad-rms-overview.aspx, Jan. 2011.

[21] A. C. Myers and B. Liskov. Protecting Privacy Using the Decentralized Label Model. *ACM Trans. Softw. Eng. Methodol.*, 9, 2000.

[22] N. Neelakantam, D. R. Ditzel, and C. Zilles. A Real System Evaluation of Hardware Atomicity for Software Speculation. In *ASPLOS*, 2010.

[23] M. Piotrowski and A. D. Joseph. Virtics: A System for Privilege Separation of Legacy Desktop Applications. Technical Report UCB/EECS-2010-70, UC Berkeley, May 2010.

[24] QubesOS: Architecture. http://qubes-os.org/Architecture.html.

[25] I. Roy, D. E. Porter, M. D. Bond, K. S. McKinley, and E. Witchel. Laminar: Practical Fine-Grained Decentralized Information Flow Control. In *PLDI*, 2009.

[26] A. Slowinska and H. Bos. Pointless Tainting?: Evaluating the Practicality of Pointer Tainting. In *EuroSys*, 2009.

[27] R. Ta-Min, L. Litty, and D. Lie. Splitting Interfaces: Making Trust Between Applications and Operating Systems Configurable. In *OSDI*, 2006.

[28] G. Venkataramani, I. Doudalis, Y. Solihin, and M. Prvulovic. FlexiTaint: A Programmable Accelerator for Dynamic Taint Propagation. In *HPCA*, 2008.

[29] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *OSDI*, 2002.

[30] A. Wolfe. Intel CTO Envisions On-Chip Data Centers. http://www.informationweek.com/news/global-cio/interviews/showArticle.jhtml?articleID=221900325, Nov. 2009.

[31] W. Xu, S. Bhatkar, and R. Sekar. Taint-Enhanced Policy Enforcement: A Practical Approach to Defeat a Wide Range of Attacks. In *USENIX Security*, 2006.

[32] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making Information Flow Explicit in HiStar. In *OSDI*, 2006.

[33] Q. Zhang, J. McCullough, J. Ma, N. Schear, M. Vrable, A. Vahdat, A. C. Snoeren, G. M. Voelker, and S. Savage. Neon: System Support for Derived Data Management. In *VEE*, 2010.