Invited Paper

pISSN 1976-913X
eISSN 2092-805X

J Inf Process Syst, Vol.9, No.2, June 2013 http://dx.doi.org/10.3745/JIPS.2013.9.2.189

The Confinement Problem: 40 Years Later

Alex Crowell*, Beng Heng Ng*, Earlence Fernandes* and Atul Prakash*

Abstract—The confinement problem was first noted four decades ago. Since then, a huge amount of efforts have been spent on defining and mitigating the problem. The evolution of technologies from traditional operating systems to mobile and cloud computing brings about new security challenges. It is perhaps timely that we review the work that has been done. We discuss the foundational principles from classical works, as well as the efforts towards solving the confinement problem in three domains: operating systems, mobile computing, and cloud computing. While common issues exist across all three domains, unique challenges arise for each of them, which we discuss.

Keywords—Confinement Problem, Covert Channels, Virtualization, Isolation, Taint Tracking

1. INTRODUCTION

In 1973, Lampson [1] first notes the confinement problem, which involves confining a program's execution to prevent it from leaking sensitive information to another program except its caller. Since then, there have been numerous works demonstrating that the confinement problem is far from solved, particularly with respect to covert channels [2-5].

To illustrate the confinement problem, suppose a bank provides banking services to its clients through customer service officers (CSO). A CSO needs to provide two security functions. First, he/she must ensure that a client cannot access information about other clients, i.e., when the CSO is accessing information on behalf of a client, it must only access information that the client is authorized to access. Second, a CSO must not disclose a client's information to another client who is not authorized to view it. One can thus view confinement as a mechanism for adhering to the least privilege principle described by Saltzer and Schroeder [6], who stated that "every program and every user of the system should operate using the least set of privileges necessary to complete the job". In the context of a typical operating system, a bank corresponds to an operating system that stores and processes sensitive information, a CSO corresponds to a process, and a client corresponds to a process owner, which can be another process or a human being.

In this paper, we re-visit the foundational principles towards solving the confinement problem, and discuss the efforts towards solving the problem in three domains: operating systems, mobile computing, and cloud computing. The paper is organized as follows. In Section 2, we present classical works examining the confinement problem. In Section 3, we discuss information flow

Manuscript received May 21, 2013; accepted June 4, 2013.
Corresponding Author: Atul Prakash
* Dept. of Computer Science and Engineering, University of Michigan, Ann Arbor, Michigan, USA (crowella@umich.edu, bengheng@eecs.umich.edu, earlence@umich.edu, aprakash@umich.edu)

control models. In Sections 4, 5, and 6, we examine the works done towards solving the problem in operating systems, mobile computing, and cloud computing respectively. We conclude in Section 7.

2. FOUNDATIONAL PRINCIPLES

The confinement problem is to prevent exfiltration of sensitive information. In attempting to characterize the confinement problem, Lampson lists seven methods by which information leaks can occur.

1. A process that is able to store and return data to the caller at a later time, i.e., a process that has memory, can leak information.
2. A process can write data to a permanent file that can be subsequently read by the process's caller.
3. A process can write data to a temporary file that can be subsequently read by the process's caller. Alternatively, a process can leak information to its caller if it is able to create a temporary file and the caller is able to test for existence of that temporary file.
4. A process can send a message using inter-process communication to another process.
5. If a process owner will be billed for the resources consumed by the process, information may be leaked by being encoded into the bills.
6. Synchronization objects such as a shared boolean variable can be used to transmit information between two processes.
7. The ratio of computing to input/output or its paging rate can be used to transmit information, albeit this method is noisy and will have low data rate.

To achieve confinement, Lampson notes that total isolation is sufficient. In total isolation, a process is not allowed to call other processes, including system calls, and its execution must not be observable. This implies that the process does not output any results and does not require any resource such as processor, storage, and network. Clearly, this rule is overly restrictive and impractical. For example, in most practical scenarios, two processes share a single file system, allowing them to test for existence of specific files as a communication means.

To relax the total isolation rule, one can postulate that if every process called by a confined process is also confined, then no leakage can occur. However, it is not possible to confine the operating system. Thus, further relaxation of the rule is necessary. Lampson suggests the transitivity rule that states that if a confined process calls another untrusted process, then the untrusted process must also be confined. This implies that we require the operating system to be trustworthy. Unfortunately, writing a trustworthy operating system is hard, since the number of possible channels for leaking information is large, although not infinite. To solve the confinement problem, it is necessary to identify these channels and ensure that they are blocked.

2.1 Covert Channels

Unfortunately, the channels can sometimes be obscure and may even involve collaboration between different entities. Not all of them can be determined easily using traditional code analysis methods. Lampson proposes three categories of channels: storage, legitimate, and covert.

Storage channels refer to disk, memory, and cache. Legitimate channels include the bill for the resources used by the process, and covert channels refer to those that are not designed for information transfer, such as CPU or memory usage. Kemmerer suggests characterizing the channels as either overt or covert instead [7]. In Kemmerer's definition, an overt channel is one that stores information in data objects such as buffers, files, and I/O devices, while a covert channel uses non-data objects, such as file locks, device busy flags, and other observable states of a system. Thus, covert channels include storage and timing channels, differing from Lampson's original note. However, Kemmerer's definition is challenged by Wray, who argues that channels cannot be cleanly categorized as storage or timing channels, as they have aspects of both [8].

Lampson proposes that the masking principle is sufficient to block all legitimate and covert channels. The principle states that the caller of a confined process must be able to distinguish the process's inputs into legitimate and covert channels.

Lipner elaborates on the confinement problem and discusses methods for enumerating storage and legitimate channels from a policy and modeling perspective [9]. We discuss one of the most commonly applied models in Section 3. In the same paper, Lipner emphasizes the difficulties in enumerating covert channels. He notes that unless the models can capture all information flows, thus allowing accesses to be controlled via policies, covert channels can occur. Lipner also highlights that all processes have the means to extract timing information, thus resulting in timing channels. To close off timing channels, each process must be provided with a virtual time that solely depends on its own activities. For example, the time needed to execute a certain function must remain constant, i.e., not influenced by other processes. Unfortunately, this is hard to achieve since resource sharing occurs on almost every system. Besides, each user has a perception of real time outside the system.

In an example of a covert channel, Shah et al. proposed Keyboard JitterBug, a keystroke logger that varies the times at which keyboard events are sent to the host [10]. A receiver analyzes the variations by monitoring the host's network traffic whenever there is an interactive network connection, thereby recovering the leaked data.

Kemmerer noted that most methods proposed for discovering channels, such as [1, 9, 11-14], are ad-hoc [7]. Thus, one cannot be sure if all storage and timing channels are discovered. Kemmerer proposed the Shared Resource Matrix methodology, which aims to provide better assurance (without guarantee) that all channels have been found. The methodology first enumerates all shared resources that can be accessed by a subject. One then determines all operation primitives of the system to generate the column headings of the Shared Resource Matrix, which is used for analyzing for presence of the channels.

A yet more subtle form of attack is the side-channel attack. Kocher [15] demonstrates how critical information used for cryptographic algorithms can be extracted using timing attacks, i.e., by observing the times needed to perform certain operations, one can deduce the information being processed. Another example of a side-channel attack is differential power analysis [16, 17].

3. INFORMATION FLOW CONTROL MODELS

Several models have been proposed towards protecting sensitive information in Multi-Level Security (MLS) systems [18]. These models are also referred to as information flow control

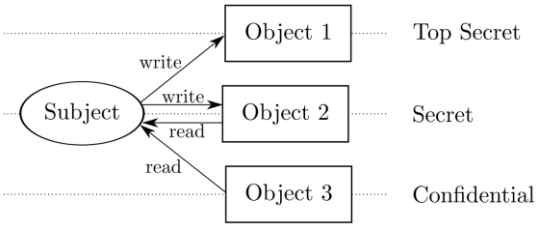

Fig. 1. An example showing that a subject can only write to objects at a higher or equal security level, i.e., Object and 2, and can only read from objects at a lower or equal security level, i.e., Object 2 and 3

models [19, 20], which differ from access control matrix models. Access control matrix models do not capture the security sensitivity of information in the objects. On the other hand, Chow et al. note that information flow control is concerned with how information is disseminated or propagated from one object to another [21].

In the information flow control models, the entities of a computer system are characterized as either subjects or objects. A subject is granted an access right as determined by a security policy to access an object [22]. The Bell-LaPadula model [23] is one of the most commonly used models, and we describe it briefly. In the model, security levels are ordered linearly, e.g., top secret, secret, confidential, and unclassified, where top secret corresponds to the highest security level and unclassified corresponds to the lowest security level. Each subject is associated with a clearance level, while each object is assigned a security classification. The Bell-LaPadula model has the following properties, which are illustrated in an example in Figure 1, where a subject that has a clearance level of secret can only write to objects classified as top secret or secret, and can only read from objects classified as secret or confidential.

- **Simple Security Property**: Also referred to as "no read up". A subject can read an object if and only if the subject has clearance level that is equal to or higher than the security classification of the object.
- ***-property**: Also referred to as "no write down". A subject can write to an object if and only if the subject has a clearance level that is equal to or lower than the security classification of the object.

Myers and Liskov propose using a decentralized model for information flow control [24]. In their work, they assume that the mechanism is reliable and efficient, and there exists a trusted execution platform functioning as trusted agents responsible for classifying/declassifying information. Zeldovich et al. demonstrate that in the absence of the notion of superuser, one can still achieve Unix-like information flow control in their work on HiStar [25].

4. Operating Systems

Actually achieving information confinement on commodity devices has been a topic of much research interest for decades. Some techniques that have been applied include the employment of information flow control features in the operating system, at different levels and in different

ways. These include mechanisms that operate at process granularity and those that operate at a finer granularity on data objects within individual processes (i.e., taint tracking). Virtualization is another tool that has been used to provide some level of isolation between processes based on their security contexts. In both cases, the handling of covert channels is still a very challenging task, and although many solutions do not even bother to address the issue, some have had varying degrees of success with addressing it. We discuss a selection of work from each of these categories below.

4.1 Information Flow Control

Although somewhat outside the scope of information flow control, one notable system for achieving some level of confinement is SELinux [26], which allows the definition of strict policies for what objects users and processes are permitted access to. The purpose of these policies is more as a measure for containing faults rather than containing sensitive data, but the same principles apply, since in both cases the goal is to prevent something from leaking to other components of the system.

SELinux is an extension to the Linux kernel, incorporated into its source in 2003, that enables mandatory access control policies on file, device, and other types of access on the system. Hypothetically, a user who wishes to enforce containment of sensitive data could implement a policy on that data that restricts access to it to only those processes the user intends to feed it to as input. The user would then have to define policies for those processes that prevent them from leaking the data once it has been read as input; for example, access to network cards and IPC objects such as domain sockets could be denied.

In this way, SELinux enables something close to what Lampson described for the masking principle, where the user describes which storage and legitimate channels may be accessed explicitly. It is important to note however, that this approach does not make covert channels explicit, and so they are not addressed at all. It is also very cumbersome to actually define such a policy that is capable of reliably restricting the sensitive data, and requires deep knowledge that only an expert is likely to possess. Furthermore, in the event the operating system is compromised, an attacker operating from the kernel level could completely subvert any policies in place, making them effectively useless. In these ways, SELinux does not serve as a practical information confinement solution for an average user. Nonetheless, its incorporation into the Linux kernel makes it freely available and easy to incorporate into real-world systems.

The HiStar operating system [25] is another, more integrated approach to directly addressing information flow control. In HiStar, information flow is built into the operating system primitives, with a labeling system that defines how information is allowed to flow between files, threads, and other system entities. One advantage of the design of HiStar is that most of the kernel is implemented within the HiStar model, untrusted and running in user mode. Since only a small, vital part of the kernel is considered trusted, the attack surface in HiStar is greatly reduced from that of systems like SELinux, where the entire Linux kernel must be trusted.

One interesting feature of HiStar is its flow permissions labeling system, which was adopted from earlier work on the Asbestos operating system [27]. Under the labeling system, permissions can be defined in arbitrary “categories” by threads, which can be used to represent any conceivable permission, and one of 4 levels of privilege can be assigned to a permission, roughly corresponding to the four permutations of read-write privileges, as well as a fifth distinct

owner's privilege, which naturally grants read/write permission plus the ability to designate privileges to others for that category. Under this permission model, as illustrated in Figure 1, the general principle that manages information flow is that data must only flow from an object with one set of permissions to another object with at least that same set of permissions.

In contrast to SELinux, storage-based covert channels are well addressed in the kernel design of HiStar. By modeling things as low-level as virtual memory and threads, and subsequently enforcing flow controls on these primitives, obvious covert channels are eliminated. HiStar does still suffer from timing-based and potentially other types of covert channels, which fall entirely outside the model devised for the operating system. Another much bigger drawback is that setting permissions in HiStar is still a laborious effort for both developers and users as in SELinux. Developers must consciously design their software in modules that allow for the safe flow of data under reasonable permission settings. Furthermore, users must overcome the high learning curve of understanding the labeling system and then set permissions for each new application and file they create. Compounding these problems is the fact that HiStar is not fully UNIX compatible, and so software may need to be ported or rewritten to achieve compatibility with this unique environment.

4.2 Virtualization

Virtualization is another technology that has come to be seen as a possible security tool in addition to its usefulness for resource sharing. By providing a high degree of isolation between individual environments, containment of an entire operating system and all processes beneath it suddenly becomes much more possible than in a traditional single OS environment.

One work that makes use of virtualization towards confinement is the Terra virtual machine platform [28]. Terra is a virtual machine monitor (VMM), that is, software that manages multiple virtual machines, in order to provide isolation and privacy between them. What distinguishes Terra from normal VMMs is that there are two defined types of virtual machines that can run on Terra: "open boxes," which have no real distinction with traditional VMs, and "closed boxes," that are highly isolated to ensure privacy and integrity of their contents. The Trusted Platform Module [29] hardware is used to ensure isolation of the closed VMs from all other VMs.

For a user on a machine running Terra, they would enforce isolation by performing different tasks on different VMs. Casual tasks that require no privacy such as simply browsing the web can be done in insecure open boxes, and the user can then switch to a closed box to perform sensitive computations or view and edit sensitive files. Although this is a good means of achieving actual containment of data, the strict isolation between VMs can also act as a hindrance to use. For example, a user who wants to download a file from over the Internet to a closed box may be unable to do so assuming Internet access is restricted, as it should be if confinement is a priority. But there shouldn't any harm in this; it could easily be a perfectly legitimate action by a normal user. This leads to another weakness of Terra, which is its vulnerability to malicious software. Although the startup process is verified via the Trusted Platform Module, software that subsequently runs on the guest is not, opening the possibility of a malicious inside program leaking data out of the machine.

In a similar fashion, QubesOS [30] serves as a hybrid VMM and operating system, managing processes and providing each one with its own virtualized view of the underlying operating system. The main difference with Terra is that processes from different VMs can be viewed on a

single screen as if they were operating in the same environment; they are all running the same operating system, although with different views of it. Faults are contained within the environment in which they occur, allowing for similar isolation to that achieved in Terra, and the Trusted Platform Module is also used to provide authenticated boot. But this similarity also implies some of the same weaknesses as found in Terra. Namely, the focus is on preventing a compromise that would have negative consequences, although no measures are taken to reduce the impact of such a consequence if it were to occur. QubesOS does however provide a mechanism for transferring files between security domains on the machine, partially improving the usability drawbacks of such strong isolation.

Storage Capsules [31, 32] aims to adapt virtualization-based solutions like Terra to a more convenient interface for users. Namely, a VMM manages a single untrusted guest operating system, which can switch between secure and insecure modes in order to gain and lose access to sensitive data, which is stored in a secure VM with a communication channel to the guest VM. The Capsules architecture is illustrated in Figure 2. A VM in Capsules begins in insecure mode, with normal access to the network and typical machine features. Upon entering a key sequence, the VMM takes a snapshot of the machine and restricts network and other device access, putting the VM into secure mode, and enabling access to secure data. When the user is finished modifying their secure data, the key sequence may be entered again to lose access to the data, restore the snapshot of machine state taken earlier during the initial transition, and regaining unrestricted access to network, storage, and other devices on the machine.

One important point worth noting is that even in the event of a compromise of the operating system, Capsules is designed to still prevent the leak of data. Aside from covert channels, only a

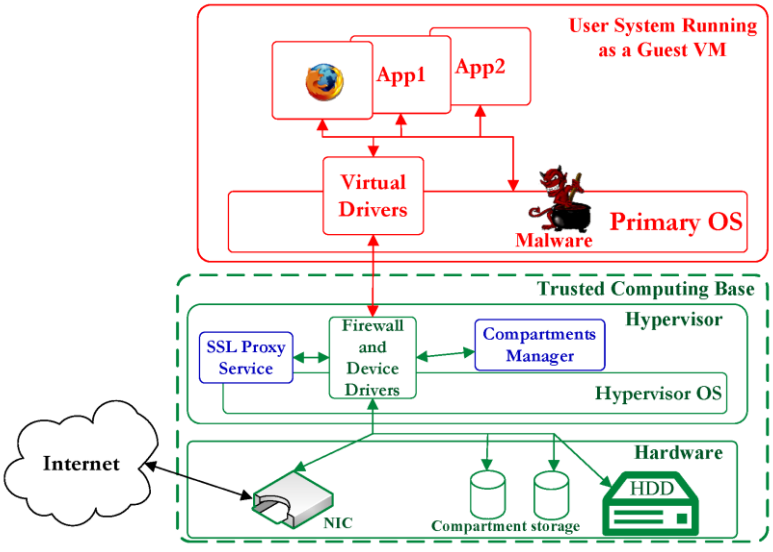

Fig. 2. In Capsules, a user's operating system runs in a virtual machine, managed by a trusted VMM. An additional (trusted) secure virtual machine is responsible for providing an interface to access secure data, stored in encrypted capsules, when the primary operating system is in secure mode. Under the Capsules threat model, the primary operating system is entirely untrusted. Figure is taken from [32].

compromise of the VMM is capable of enabling this, making it more robust than the previously mentioned systems. Capsules also simplifies usability while maintaining strong isolation, in that a user can still access the files available to them in insecure mode (although they cannot write to them, as this would be an obvious path for leaking data) and are only unable to access the secure data outside of secure mode.

Covert channels are discussed in great detail for Capsules in [31, 32], with many different storage and timing-based channels considered, and suggestions made for how they might be mitigated. The overhead over a normal system was also measured, and they found as much as a 38% overhead for a simple test build of the Apache web server, but lower overheads for most other tasks. Additionally, they observed transition delays on the order of seconds both to and from the secure mode, resulting from the taking of snapshots upon entry and their restoration upon exit. Improvements in the transition times have been observed in later implementations.

Secure Data Capsules, not to be confused with the Storage Capsules described earlier, is a proposal by Maniatis et al. that describes the conceptual architecture of secure data elements cryptographically protected and bundled with provenance and access control policies [33]. Their concept invokes the use of virtualization as a means of minimizing the trusted computing base by placing the isolation system below the operating system, and taint tracking is invoked as the solution to achieving fine-granularity knowledge of how the data is being used, for example whether it is about to be sent over the network. They discuss the many challenges that need to be overcome to achieve their vision, and acknowledge that covert channels are likely infeasible to prevent.

5. MOBILE COMPUTING

Mobile phones are the new computing revolution. They have progressed from simple devices the size of a brick that only made phone calls to wafer thin computing platforms that are capable of everything from making phone calls to reading email. Hence, it should come as no surprise that these devices contain a wealth of confidential and personal information. For example, use of mobile apps to perform mobile banking is common. The device possesses information of financial value. Mobile apps also manage confidential emails in the Bring-Your-Own-Device (BYOD) setting. Thus the informational content of these devices is high. These devices are vastly different than desktops that are used to perform these computational tasks. Smartphones are more integrated with everyday life. People carry them everywhere and they have a multitude of sensors to perceive the environment. Smartphones also include many information-sharing channels like bluetooth, NFC and Wi-Fi. More importantly, several covert channels have been discovered as well [34] that allow a device to infer keystrokes made by a user with the help of the device orientation. Keystroke data is sensitive as the user may be typing Internet banking passwords. Therefore, it is apparent that the study of information confinement in the context of smartphones is an interesting and challenging subject. In the coming sections, we explore several important results from the literature.

5.1 Classical Information Flow Control

A popular and effective technique to govern (and hence confine) information flow is *taint propagation* (also known as dynamic taint analysis) [35]. This method involves associating

metadata with a program variable, often referred to as a taint value. The taint value can be as simple or as complex a designer wishes it to be. However, adding complexity always has negative side-effects in terms of memory overhead. These taint values are then propagated throughout the program, always tracking the variable it is associated with. For example, suppose the taint value for a program variable *x* is initialized to the value 1. This could signify that *x* contains sensitive data. Now *x* is assigned to another variable *y*. The taint of *x* will be transferred to *y*. The rules of taint transfer are governed by taint propagation rules. A taint transfer is effected only when the corresponding variables' contents flow from one to another, possibly through transformations. The strength of taint propagation is that it is capable of tracking the taintedness of a variable through these transformations, including cryptographic operations. The use of this method in confining information is immediately obvious. If you can track something, you can confine its movements by imposing policies on how it flows.

Enck et al. propose the TaintDroid system [36], which applies taint propagation to the Android platform. TaintDroid proposes dynamic taint analysis to control how data flows between applications. TaintDroid is capable of tracking sources of specific tainted data. In TaintDroid, taints are statically associated with predefined data sources, such as the contact book, SMS messages, the phone number, the device identifier (IMEI), etc. TaintDroid limits the flow of tainted data by tracking the taints in the outbound network connections. However, TaintDroid is not capable of enforcing separation of operation modes. For instance, TaintDroid would treat private and work contacts as the same type (because they are tainted with the same taint) applying the same policy. Therefore it is not possible to have in TaintDroid corporate applications that can only access corporate data. The same holds true for private applications.

Similar conclusions can be drawn for Paranoid Android [37]. Paranoid Android proposes tainting of data for runtime checks. In Paranoid Android security analysis is executed by a trusted remote server, which hosts the replicas of smart phones in virtual environments. However, this approach has a severe impact on the device performance since execution traces have to be continuously sent to the remote servers.

5.2 Inlined Reference Monitors

Inlined Reference Monitors (IRM), introduced in [38], are reference monitors that are interspersed with application code. A reference monitor is an entity that regulates access to resources. An example is the Android permission reference monitor. Whenever an application tries to read the location co-ordinates, for example, an implicit check is performed that verifies that the application has the requisite permission. This monitor is outside the address space of the application. An inlined monitor is inserted within the application code. Thus, it does not require changes to the system code to implement security solutions. However, care must be taken that the application cannot subvert the mechanism by corrupting the code sections belonging to the monitor.

Aurasium [39] is a system by Xu et al that uses IRMs to implement security policies for the Android platform. The reference monitors are inserted via bytecode rewriting. On the Android platform, every high level framework service invoked is through Inter-process Communication (IPC). Thus, if an application requests for contacts data, it will take the form of an IPC request. Aurasium intercepts these IPC requests at the *libc* level and infers what high level action is being performed. It then makes a decision whether to allow the action or not. In the case of data

related actions, this becomes an information confinement solution. By interposing monitors on actions that access and disseminate information, control over its flow is achieved. However, it is worth noting that this method is not as fine grained as taint propagation. For example, it can only perform a security action for known information accesses. This means that if an application requests contacts data, the IRM can impose a policy at this point. If the policy accepted the action, there is nothing to prevent the application from transmitting this data over the network. One may say that a network access policy is useful, but we point out that the IRM will not be able to distinguish data at the network interface.

5.3 Lightweight Virtualization

Providing data and execution isolation through virtualization is a very effective technique. Virtualization on desktop computers has existed for quite some time and it is considered a mostly solved problem. However, virtualization on mobile devices is a subject of research. The small form factor and limited battery life impose new usability restrictions. For example, simply having a window on a desktop with the whole UI of the virtualized system contained within suffices. Such a model however, does not work for smartphone screens. Just imagine having to use a virtualized system contained within a small window on an already (relatively) tiny screen. Another issue is that of power consumption. Today's devices run sophisticated Operating Systems with a multitude of subsystems. Add to that, another complete (or possibly more than one) operating system, and battery life is comparable to a sand dune. This has motivated research into lighter weight virtualization systems.

Russello et al. introduce MOSES [40], which is a lightweight virtualization mechanism. The system aims to tackle the problem of confining information to various operating modes. An application can be in different operating modes at various times. For example, while at work, the Email application can be in "work" mode. While at home, it can be in "personal mode" allowing access to personal emails and not work related emails. This mode of use may sound familiar and is commonly known as the Bring-Your-Own-Device paradigm. In such settings, personal devices are allowed access to confidential company secrets while an employee is at work. Any compromise of the device will directly compromise company data. Thus, strong information confinement guarantees are needed. MOSES uses a combination of taint propagation, reference monitors and filesystem namespaces to achieve isolation in a lightweight manner. Policies are used to specify different operating modes. The policies control the behavior of the reference monitors and taint trackers. For example, an Android application like Email, which stores emails in a database file, will have different views of the filesystem depending upon its operating mode. Furthermore, the policies (and hence operating modes) can be configured to switch automatically based on the physical context of the device. This is a form of access control known and contextual access control and we detail its applicability to information confinement in Section 5.4.

Cells [41] by Andrus et al. proposes a lightweight virtual smartphone architecture for Android. A virtual-phone is the unit of abstraction and it is basically a complete Android user space. Cells is based on the concept of namespaces, which is also utilized by MOSES. Specifically, Cell introduces the concept of device namespaces. These are namespaces for common Android devices such as the GPS, binder, and ashmem, among others. Each device driver is modified to maintain context information. Thus it has its own context when one virtual-phone is running. Cells also uses the recently introduced (in the linux kernel) UID namespace mechanism wherein processes

may have uid 0, but may not possess superuser privileges outside the context of its virtual-phone. Hence, Cells provides the same illusion as traditional desktop virtualization and has similar information confinement guarantees. Information from one virtual-phone cannot leak to another. However, any vulnerability in a program that needs real superuser privileges (for example, the volume daemon) running in a virtual-phone will compromise the base kernel. This is a fundamental difference between Cells and traditional virtualization mechanisms that contain separate kernels.

5.4 Contextual Information Confinement

Often, the context in which information is accessed is important. For example, a company may not want an employee to access work-related information by keying in passwords in a crowded place like a subway. There is a threat of information disclosure through channels such as shoulder surfing. Another example is that a person may not want to use Internet banking on public Wi-Fi. Such a form of access information control, which is based on the physical context of the device, is also referred to as *contextual access control.* A context may be defined as a collection of physical environmental factors that determine the state of the device. For example, GPS co-ordinates define the positional context of a device. Similarly, the accelerometer defines the orientation context of a device.

Context Related Policy Enforcement (CRePE) for Android [42] is the first system that implements contextual control for Android smartphones. The basic idea in such forms of information confinement is that the reference monitor is made context aware through the use of context aware policies. In CRePE, policies include an additional context component that indicates under what environmental situations the policy is to be considered active. For example, if the policy states that the camera can be accessed only within a range of 100m of some specified geographical co-ordinates, then any attempt to access the camera outside this range will not be allowed. Hence, one can imagine policies that confine when company related data is accessed. A logical choice would be to have a policy that allows the access of this information only when the device is physically located within the company premises and connected to a secure wireless network.

6. CLOUD COMPUTING

With cloud computing becoming commonplace, we can expect the confinement problem to exacerbate in the cloud. While numerous threats exist for the cloud as described by Vaquero et al. [43], we are primarily concerned with information leakage. Ristenpart et al. show how one can discover the internal cloud infrastructure for Amazon EC2, and estimate the location of a target VM [44]. The attacker can then instantiate new VMs on the same machine as the target to launch cross-VM side-channel attacks to extract information from the target.

Besides inheriting the security issues commonly associated with individual components of the cloud, new issues arise. For example, since almost every resource, e.g., disk, memory, network, etc., is shared, the possibility for covert channels increases. As Aviram et al. puts it [45], timing channels is a security challenge in cloud computing because (i) massive parallelism making timing channels pervasive and hard to control, (ii) users can steal information from other users without leaving detectable trails, (iii) only the cloud provider has the means to detect and report

the attacks, although they are usually not incentivized to do so, (iv) resource partitioning can limit the statistical sharing efficiency that underpins the motivation for cloud computing.

The threat model has also changed from one where the hardware, operating system, and installed applications can be trusted to one where these entities may be malicious. As examples of threats in the cloud, Mulazzani et al. examined cloud storage and found that Dropbox, a popular cloud storage service, is used to store copyright-protected files, and can be used to hide files in online slack space [46]. Wu et al. discussed an attack on cloud computing capable of high bandwidth and reliable data transmission in the cloud [47]. They improved on attacks using cache channels by proposing a pure timing-based data transmission scheme. They also exploited the memory bus to achieve the high bandwidth.

Perez et al. propose virtualizing the Trusted Platform Module in their work on vTPM [48], thus allowing trusted computing for an unlimited number of virtual machines on a single hardware platform. Towards solving the confinement problem in the cloud, Keller et al. note that most of the security issues in the cloud are because of the virtualization layer, i.e., sharing of resources [49]. Thus, they propose removing the virtualization layer, and suggest a NoHype architecture to achieve the same features required for virtualization using currently available hardware extensions to processors and I/O devices.

Aviram et al. suggest using provider-enforced deterministic execution instead of resource partitioning to prevent computing the results of a task from being dependent on the execution timing [45].

Zhang et al. note that most clouds are implemented using commodity virtualized infrastructures, thus attacks can lead to leakage of sensitive data [50]. In their work on CloudVisor, they propose separating resource management from security protection in the virtualization layer. A security monitor is placed below a commodity virtual machine monitor using nested virtualization to protect the hosted VMs.

In the future, we believe more efforts will be invested towards providing security assurance in the applications provided by the cloud providers. One possible approach to achieve this may be to leverage hardware, which provides a small trusted computing base. While this may not eliminate issues such as supply chain attacks, the bar for attacks by cloud providers is raised.

7. CONCLUSION

The confinement problem has led to numerous security issues, particularly covert channels. With almost every piece of information being digitized, mitigating the confinement problem, and thus preventing security issues, such as information leakages, becomes more critical than before. With the huge amount of efforts invested towards solving the problem, it is important that we review the works that have been done four decades after the problem was first defined formally, thus paving the way forward. We began our discussions of the confinement problem by providing a review of the classical works. Following that, we examined the confinement problem and efforts to solve it in three domains: operating system, mobile computing, and cloud computing.

REFERENCES

[1] Butler W. Lampson. A note on the confinement problem. Commun. ACM, 16(10):613-615, October 1973.

[2] Zhenghong Wang and Ruby B. Lee. Covert and Side Channels Due to Processor Architecture. In Proceedings of the 22nd Annual Computer Security Applications Conference, ACSAC '06, Washington, DC, USA, 2006. IEEE Computer Society, pp.473-482.

[3] Colin Percival. Cache missing for fun and profit. In Proc. of BSDCan 2005, 2005.

[4] Onur Aciicmez. Yet another MicroArchitectural Attack: exploiting ICache. In Proceedings of the 2007 ACM workshop on Computer security architecture, CSAW '07, New York, NY, USA, 2007. ACM, pp.11-18.

[5] Onur Aciicmez, Cetin Kaya Koc, and Jean-Pierre Seifert. Predicting secret keys via branch prediction. In Proceedings of the 7th Cryptographers' track at the RSA conference on Topics in Cryptology, CT-RSA'07, Berlin, Heidelberg, 2006. Springer-Verlag, pp.225-242.

[6] Jerome H. Saltzer and Michael D. Schroeder. The Protection of Information in Computer Systems. 1975.

[7] Richard A. Kemmerer. An Approach to Identifying Storage and Timing Channels. In ACM Transactions on Computer Systems, 1983.

[8] J.C. Wray. An analysis of covert timing channels. In Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on, 1991, pp.2-7.

[9] Steven B. Lipner. A comment on the confinement problem. In Proceedings of the fifth ACM symposium on Operating systems principles, SOSP '75, New York, NY, USA, 1975. ACM, pp.192-196.

[10] Gaurav Shah, Andres Molina, and Matt Blaze. Keyboards and covert channels. In Proceedings of the 2006 USENIX Security Symposium (July-August), 2006, pp.59-75.

[11] Jonathan K. Millen. Security Kernel validation in practice. Commun. ACM, 19(5):243-250, May 1976.

[12] Marvin Schaefer, Barry Gold, Richard Linde, and John Scheid. Program confinement in KVM/370. In Proceedings of the 1977 annual conference, ACM '77, New York, NY, USA, 1977. ACM, pp.404-410.

[13] Stanley R. Ames and Jonathan K. Millen. Interface verification for a security kernel. Technical report, Infotech International, Ltd., Maidenhead, Berkshire, UK, 1978.

[14] C. Kline. Data security: Security, protection, confinement, covert channels. PhD thesis, UCLA, 1980.

[15] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '96, London, UK, UK, 1996. Springer-Verlag, pp.104-113.

[16] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In CRYPTO, 1999, pp.388-397.

[17] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. J. Cryptographic Engineering, 1(1):5-27, 2011.

[18] Hossein Bidgoli. Handbook of Information Security, Threats, Vulnerabilities, Prevention, Detection, and Management, Vol.3. John Wiley & Sons, 2006.

[19] John McLean. Security models and information flow. In Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on. IEEE, 1990, pp.180-187.

[20] Andrew S Tanenbaum and Maarten Van Steen. Distributed systems, Vol.2. Prentice Hall, 2002.

[21] Randy Chow and Theodore Johnson. Distributed Operating Systems and Algorithms. Addison Wesley, 1997.

[22] Butler W. Lampson. Protection. In Princeton University, 1971, pp.437-443.

[23] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Mathematical Foundations. Technical Report MTR-2547, Vol.1, MITRE Corp., Bedford, MA, 1973.

[24] Andrew C. Myers and Barbara Liskov. A Decentralized Model for Information Flow Control. In Proc. 17th ACM Symp. on Operating System Principles (SOSP, 1997, pp.129-142.

[25] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazi`eres. Making information flow explicit in HiStar. In Proceedings of the 7th USENIX Symposium on Operating Systems Design

and Implementation- Vol.7, OSDI '06, Berkeley, CA, USA, 2006. USENIX Association, pp.19-19.

[26] National Security Agency. Security Enhanced Linux, May 2013.

[27] Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazieres, Frans Kaashoek, and Robert Morris. Labels and event processes in the Asbestos operating system. ACM SIGOPS Operating Systems Review, 39(5):17-30, 2005.

[28] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A virtual machine-based platform for trusted computing. In ACM SIGOPS Operating Systems Review, Vol.37. ACM, 2003, pp.193-206.

[29] Andrew Martin et al. The ten page introduction to trusted computing, 2008.

[30] Joanna Rutkowska and Rafal Wojtczuk. Qubes OS architecture. Invisible Things Lab, Tech. Rep, 2010.

[31] Kevin Borders, Eric Vander Weele, Billy Lau, and Atul Prakash. Protect ing Confidential Data on Personal Computers with Storage Capsules. In USENIX Security Symposium, 2009, pp.367-382.

[32] Billy Lau, Atul Prakash, and Venkatanathan Annamalai. Accessing Trusted Web Sites from Low-Integrity Systems without End-Host Snooping. In SocialCom/PASSAT, 2011, pp.1012-1019.

[33] Petros Maniatis, Devdatta Akhawe, Kevin Fall, Elaine Shi, Stephen McCamant, and Dawn Song. Do you know where your data are?: secure data capsules for deployable data protection. In Proceedings of the 13th USENIX conference on Hot topics in operating systems, HotOS'13, Berkeley, CA, USA, 2011. USENIX Association, pp.22-22.

[34] Liang Cai and Hao Chen. TouchLogger: inferring keystrokes on touch screen from smartphone motion. In Proceedings of the 6th USENIX conference on Hot topics in security, HotSec'11, Berkeley, CA, USA, 2011. USENIX Association, pp.9-9.

[35] Schwartz, Edward J. and Avgerinos, Thanassis and Brumley, David. All you ever wanted to know about Dynamic Taint Analysis and Forward Symbolic Execution. In proceedings of the 2010 IEEE Symposium on Security and Privacy, S&P '10, Washington DC, USA, 2010, pp.317-331.

[36] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In Proceedings of the 9th USENIX conference on Operating systems design and implementation, OSDI'10, Berkeley, CA, USA, 2010. USENIX Association, pp.1-6.

[37] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid Android: versatile protection for smartphones. In Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC '10, 2010, pp.347-356.

[38] U´lfar Erlingsson. The inlined reference monitor approach to security policy enforcement. PhD thesis, Ithaca, NY, USA, 2004. AAI3114521.

[39] Rubin Xu, Hassen Sa¨ıdi, and Ross Anderson. Aurasium: practical policy enforcement for Android applications. In Proceedings of the 21st USENIX conference on Security symposium, Security'12, Berkeley, CA, USA, 2012. USENIX Association, pp.27-27.

[40] Giovanni Russello, Mauro Conti, Bruno Crispo, and Earlence Fernandes. MOSES: supporting operation modes on smartphones. In Proceedings of the 17th ACM symposium on Access Control Models and Technologies, SACMAT '12, New York, NY, USA, 2012. ACM, pp.3-12.

[41] Jeremy Andrus, Christoffer Dall, Alexander Van't Hof, Oren Laadan, and Jason Nieh. Cells: a virtual mobile smartphone architecture. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP '11, New York, NY, USA, 2011. ACM, pp.173-187.

[42] Mauro Conti, Vu Thien Nga Nguyen, and Bruno Crispo. CRePE: context-related policy enforcement for android. In Proceedings of the 13th international conference on Information security, ISC'10, Berlin, Heidelberg, 2011. Springer-Verlag, pp.331-345.

[43] LuisM. Vaquero, Luis Rodero-Merino, and Daniel Morn. Locking the sky: a survey on IaaS cloud security. Computing, 91(1):93-118, 2011.

[44] 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, New York, NY, USA, 2009. ACM, pp.199-212.

[45] Amittai Aviram, Sen Hu, Bryan Ford, and Ramakrishna Gummadi. Determinating timing channels in compute clouds. In Proceedings of the 2010 ACM workshop on Cloud computing security workshop, CCSW '10, New York, NY, USA, 2010. ACM, pp.103-108.

[46] Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar Weippl. Dark clouds on the horizon: using cloud storage as attack vector and online slack space. In Proceedings of the 20th USENIX conference on Security, SEC'11, Berkeley, CA, USA, 2011. USENIX Association, pp.5-5.

[47] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In Proceedings of the 21st USENIX conference on Security symposium, Security'12, Berkeley, CA, USA, 2012. USENIX Association, pp.9-9.

[48] Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: virtualizing the trusted platform module. In Proc. 15th Conf. on USENIX Security Symposium, 2006, pp.305-320.

[49] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B. Lee. NoHype: virtualized cloud infrastructure without the virtualization. In Proceedings of the 37th annual international symposium on Computer architecture, ISCA'10, New York, NY, USA, 2010. ACM, pp.350-361.

[50] Fengzhe Zhang, Jin Chen, Haibo Chen, and Binyu Zang. CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP '11, New York, NY, USA, 2011. ACM, pp.203-216.

Alex Crowell

Alexander Crowell is a doctoral candidate in Computer Science at the University of Michigan, Ann Arbor, focusing on research in computer security and privacy under advisor Atul Prakash. He previously received a Bachelor of Science in Computer Science from Rutgers, The State University of New Jersey, and a Master of Science in Computer Science from the University of Michigan, Ann Arbor.

Beng Heng Ng

Beng Heng received his Ph.D. from the University of Michigan, and was advised by Prof. Atul Prakash. His thesis focused on techniques towards achieving the least privilege principles on various computer systems. He is also interested in security including, but not limited to, malware polymorphism, propagation, attack vectors and covert channels. Prior to pursuing his Ph.D., he worked as a security software researcher. His job scope included penetration testing, security software analysis and malware analysis

Earlence Fernandes

Earlence Fernandes graduated with a Bachelor in Computer Engineering from the University of Pune in 2009. He worked as a Scientific Programmer in the Systems Security group at Vrije Universiteit Amsterdam until 2012. Currently, he is a PhD student at the University of Michigan, Ann Arbor, USA; working in the areas of Smartphone Security and runtime optimizations for Information Flow Control.

Atul Prakash

Atul Prakash is a Professor in Computer Science and Engineering at the University of Michigan, Ann Arbor with research interests in computer security and privacy. He received a Bachelor of Technology in Electrical Engineering from IIT, Delhi, India and a Ph.D. in Computer Science from the University of California, Berkeley. His research on online web security was widely quoted, including Business Week, Barron's, and BBC Digital Planet. He has also examined the problem of privacy and security risks in emerging pervasive systems, such as online social networks. In a NIST-funded project, he explored solutions to collecting data for monitoring the health of cyber-physical systems, such as bridges. Dr. Prakash received an award for the best undergraduate project at IIT Delhi and the research excellence award at the University of Michigan. Among his earlier research work, he was one of the designers of the Upper Atmospheric Research Collaboratory (UARC) project, among the first scientific collaboratory systems, which was recognized by the Smithsonian-Computerworld for its contributions to science. Among educational activities, he helped create the interdisciplinary undergraduate program in Informatics at the Univ. of Michigan. Dr. Prakash is a member of IEEE and ACM.