Virtualization in an embedded environment

Joakim Svensson
Henrik Andersson
Lunds University

Abstract

Up until now, the security in a mobile phone has not been a very big issue. But with the increased use of computer-like smartphones, and particularly the introduction of the open-source Android operating system, it is more important to make the phone safe from malicious software, to not compromise the security of individual users. This article will give an introduction on how virtualization can help increasing the security of a mobile phone, without decreasing the performance of a device already low on resources. It will describe what virtualization is, some different types of virtualization and why para-virtualization seems to be the best method to use on a mobile phone. It will also show some things needed to be done in order to port an operating system to support the para-virtualized hypervisor Xen-ARM.

1 Background

As more complex operating systems for mobile phones are being developed, they can possibly make the phone more sensitive to malicious attacks. The recently introduced Android platform has made manufacturers more aware of this than ever. Android is built on the Linux kernel and is open source, meaning everyone may view, and rewrite, the source code. Virtualization, and especially para-virtualization, can provide a method to make the mobile phones more secure. The use of virtualization can help protecting the parts of the device that might be extra sensitive. In order to explain this concept it is to have some basic knowledge of virtualization.

2 Virtualization

Virtualization is a technique that provides a virtual interface between one or more operating systems and the hardware of a machine. It was developed by IBM in the 1960's[1], to get as much out of a mainframe computer as possible. Users could login to the mainframe and use its resources such as the CPU or the RAM. This was important since mainframes were very expensive and users needed to get as much out of them as possible. Nowadays, computers provide a lot more performance for much less money. Every user can have their own computer instead of having to login to a mainframe. Virtualization is no longer needed in the same way as before. Instead, new areas of use have been developed. Today virtualization exists in many different ways, and many users are using a virtualized environment without even knowing. For example, when executing a Java program, it is run in a virtualized environment that shares the same resources to the application regardless of which operating system it is run in. This environment is called the Sun Java Virtual Machine[2] and is used to achieve platform independency for the Java programming language.

In contrast to virtualization being hidden from the user, there are also types that are more hands-on. One example is when using an emulator to run an application that usually requires another hardware platform than the current system is using. One popular emulator is the Qemu that for instance can emulate a device with an ARM architecture, popular in mobile phones and embedded devices. Qemu makes software that normally only would run on an ARM device available to run on an ordinary PC using x86. This is very helpful when developing applications for an ARM device, but the programming is done on a x86 computer. The developer is able to start and test the software in the emulator even without the correct hardware at hand.

Another use of virtualization is when running two different operating systems at the same time. For instance, if a developer wants his software to be able to run in different operating systems, he may export it to the other systems and test it without switching computer or even rebooting to switch operating system. Virtualization can also help during bug testing. If the application causes a bug that normally would make the operating system crash, in a virtualized environment only one system would crash, the other would stay intact.

To achieve this type of virtualization, a certain kind of software needs to handle the communication between oper-

ating systems and the physical hardware of the machine. The software also needs to take responsibility of setting up the needed virtual environments regarding memory addresses, hard disk drive location, network interface, etc. This software is usually referred to as a *hypervisor*[1]. How the hypervisor works depends a lot of what kind of virtualization and which specific hypervisor that is used.

One popular hypervisor is VMware[3], commonly used to run another operating system inside an already working one, for instance running a Linux distribution within Windows. VMware makes a virtual copy of the hardware and lets the virtualized operating system see this copy. The virtualized guest will believe that this is the real machine, and communicates with it as it usually does. Whenever the virtual OS tries to communicate with hardware, VMware uses different lookup tables to translate between virtual and real hardware. This method of virtualizing, when the operating system think it is installed on the real hardware, but it rather is a virtual version of the real hardware, is called full virtualization.

When using VMware as mentioned above there are quite a large loss of performance because a lot of overhead work is needed by the hypervisor to translate between virtual and physical hardware all the time. This makes the method not very attractive to use in a mobile phone system, where the resources are low compared to a PC. One way to increase the performance and decrease the work of the hypervisor is to let the virtualized operating systems be able to communicate with the real hardware instead of a virtual copy. That way, the virtualized guest can make the calls to the hardware and the hypervisor does not need to translate them all the time. The hypervisor will, besides creating and configuring the virtual environment, only be responsible for assuring that the calls are correct. For example if one virtual guest tries to access memory that belongs to another guest, the hypervisor will deny access. This virtualization method is called para-virtualization and even though there are some drawbacks it is the best suited method to use in a mobile phone environment because of the gain in performance. One of the more popular hypervisors that implements para-virtualization is Xen and in this article its ARM derivate Xen-ARM[4] will be used as an example. This because Xen-ARM is open source, free to download and examine. It is also intended to work with the hardware on a mobile phone.

3 Security

As stated earlier, virtualization can be used in order to increase the security of a mobile phone. With Xen-ARM the running operating systems are isolated from each other. Both have access to the same hardware and share for instance network card and monitor drivers, but when it comes to the memory the hypervisor divides the RAM so that one operating system never will have access to any memory that belongs to another system[5]. However, by going through the hypervisor, information can be passed from one operating system to another. This leads to interesting cases where developers can use the hypervisor to define what information may be passed between the systems. One system can handle all interaction with the user while sending all critical data to another, secured system, minimizing the risk of malicious software accessing sensitive areas. The non-secure operating system could for instance be Android, which the user may tamper with as much as he wants, while the secure system could be a system specifically programmed for handling secure critical tasks. Figure 1 shows how this could look.

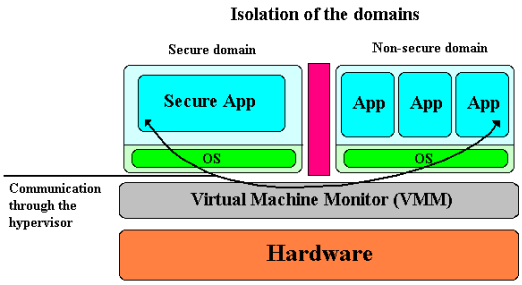

Figure 1. Isolation of the domains

This makes para-virtualization and Xen-ARM excellent candidates for creating a secure environment on a mobile phone. One disadvantage however, is that the operating system needs to be modified. What is required to be changed, and why, will be discussed soon.

4 Hypercalls

A hypercall is to the hypervisor what a system call is to an operating system, or in other words, a hypercall is a way for the virtualized operating systems to make the hypervisor handle privileged operations. Much like a system call which lets a user application communicate to the operating system that it needs to execute a command with higher priority than normal[6].

The hypercall system is an important mechanism when para-virtualizing, since the hypervisor replaces the operating system as the most privileged software. Some commands, such as memory management instructions, are required to run with the highest level of priority in the CPU. The operating system therefore needs a way to communicate with the hypervisor and this is why hypercalls was in-

[1] Another common name is Virtual Machine Monitor.

vented.

In Xen-ARM, a hypercall is basically a function added to the operating system that can be called whenever it is needed. All current implemented hypercalls sets important information in various CPU registers and then invokes a certain software interrupt. Xen-ARM is designed to trigger whenever that software interrupt is invoked and checks the registers to determine which hypercall that the OS has called. Every hypercall is mapped to a function in the hypervisor source code and the specific function is executed.

Hypercalls is used for instance when an OS needs to update the memory management unit (MMU) of the processor, to switch between foreground domains or to retrieve the system time. To report back the results of a hypercall to the OS, Xen-ARM makes use of a mechanism called event channels. An event channel is like a communication link where the hypervisor can queue different events and let the OS take the appropriate actions based on the queued events. This way, the OS can call for a MMU update; the hypervisor executes the update instructions and returns information about which memory addresses the OS can use. The OS then updates its internal memory structures with the new memory addresses and any current user process can continue to run. An example of this hypercall can be viewed in Figure 2.

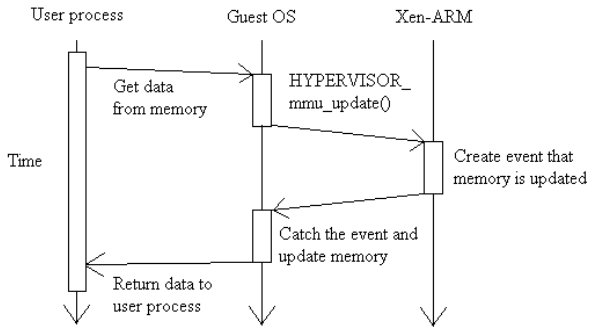

Figure 2. How hypercalls is used to for instance update the MMU.

5 Modifications of Linux

As previously stated, an OS needs to be modified to work with Xen-ARM. It is necessary for the OS to have knowledge about being virtualized, because when para-virtualizing, it will handle some hardware communication itself. There are quite a few different places to change the source code, but the most extensive changes are made to the interrupt handling system and the memory management.

When different interrupts occur in the system, Xen-ARM will trigger and find out what interrupt that has been invoked. If it is a timer interrupt or a serial interrupt, Xen-ARM has internal functions that will handle the interrupts and for instance, increase the system time structure. All other interrupts will be put in an event channel, mentioned earlier. Those interrupts will be handed back to the OS that will execute its own event handler to take care of them. To clarify, the modifications include adding support for event channels instead of handling interrupts. They also include functions to disable or enable the event channels when necessary.

The memory management of the OS is also modified. The guest will have access to read memory on addresses that is valid for that specific guest, but to write something to the memory the OS will have to go through Xen-ARM. In full virtualization, all communication is run through the VMM, which requires a lot more work since the VMM translates every virtual memory address to a real address. Xen-ARM on the other hand gives the guests access to different areas of the memory (at some occasions, guests even get to share memory) and also provides them with the real addresses. In this way, the guests get instant access to read hardware, and when writing to the memory all Xen-ARM does is to validate that the guest actually have the right to write to that specific memory.

Linux (and other operating systems) have a clever way to make use of more memory than actually is available. This is done by dividing the memory into pages, which basically are chunks of consecutive addresses. To reach a specific page, the OS goes through a chain of page tables that in the end points to actual memory (it can be RAM or a hard disk drive)[7]. Xen-ARM provides page tables for the guest operating systems, both tables that a guest can use for its individual user processes, and tables that the guest can use as machine memory. This is in contrast to a ”normal” system, where the OS have access to all real memory, and handles the page tables itself. Therefore, it is necessary to rewrite the memory management code to support the virtual machine's memory, and the memory that user processes will use.

When initializing the kernel, all the interrupts and the memory is setup, the OS also saves important information about the machine and the hardware. In the para-virtualized system however, Xen-ARM will handle a lot of the information, and provide it to the guest OS. The kernel setup procedure is therefore the third main thing that is changed in the OS source code. Instead of having the guests communicating with the hardware to find out for instance CPU ID, they will get this information directly from Xen-ARM.

There are more changes to the source code that is necessary, for instance driver code and functions that will print text to a console, but the three areas previously mentioned

is where the biggest modifications are made.

6 Conclusions

As you probably can understand it is not an easy task to virtualize the Android platform on a mobile phone. It is necessary to configure the hypervisor for the target hardware and modify the Android kernel source to be compatible with the hypervisor and the para-virtualization technique. It is also required to implement a protocol that can be used for secure communication between the multiple operating systems in the phone. On top of that the word *secure* must be defined in terms of what operations that absolutely needs to stay uncompromised (target for malicious attacks); also, one or more operating systems must be configured or implemented according to the desired security level.

However, all the extra work aside, the para-virtualization technique has potential to provide for a very secure mobile phone environment, without losing too much performance due to overhead virtualization operations. In the future, embedded CPUs might include hardware support for para-virtualization, which will remove the need to modify the operating systems, further improving the feasibility of this virtualization method.

References

[1] John Fisher-Ogden. Hardware support for efficient virtualization. Background and information about different virtualization styles, 2006.

[2] Sun java official homepage. `http://java.sun.com/`.

[3] Vmware hompage. `http://www.vmware.com`.

[4] Homepage of the xenarm project. `http://wiki.xensource.com/xenwiki/XenARM`.

[5] Rafal Wojtczuk. Subverting the xen hypervisor, 2008.

[6] hypercall article on xenwiki. `http://wiki.xensource.com/xenwiki/hypercall`.

[7] Wolfgang Mauerer. *Professional Linux Kernel Architecture*. Wrox Press Ltd., Birmingham, UK, UK, 2008.