MoVirt: More Mobile than Mobile

Will Dietz, Kevin Larson, Shivaram Venkataraman
{wdietz2, klarson5, venkata4}@illinois.edu
University of Illinois at Urbana Champaign

1 Introduction

Modern mobile devices run third party applications to perform complex tasks like web browsing, banking and gaming. Recent studies have found that smart-phones are the target of an increasing number of malware attacks [22, 4, 5] and their security is important as personal data such as contacts, credit card numbers and passwords are often stored on the device. While some security models [3] provide a stronger process level isolation among applications, operating system bugs such as [12, 11, 13] allow malicious applications to take over the device. Recent reports have even found that some smart phones had the Mariposa botnet pre-installed [27]. Virtualization can be useful for secure isolation of third party code from confidential data and provide greater defense-in-depth against attacks on the system.

In recent years, virtual machines have become prevalent in cluster computing environments [2] as they provide isolation for shared usage of machines in a data center. As a result of hardware improvements, smart phone configurations found today resemble desktop machines from few years ago and many of them run commodity operating systems. There is a growing interest in academia [24] and industry [6] about the benefits of virtualization on these devices. Virtualization provides better security guarantees in mobile devices than current solutions offer as well enabling useful applications like environment migration.

Migrating a system to a mobile device can take advantage of network or computation facilities that are closer to the user's location and provide the user with a consistent experience irrespective of the network connectivity. Environment migration has been studied earlier, in the context of servers in a cluster [23] and enables administrators of clusters to perform maintenance tasks without interruption. On the other hand, migration techniques on mobile devices can help maintain consistent snapshots which allow easy transfer of data when users switch mobile phones and to roll-back the system to a previously known state.

Existing solutions for isolation of mobile applications introduce a Type I hypervisor [29, 26] which manages the phone's hardware and allows running multiple operating system on top. While these solutions are useful, it would be better to have isolated applications which are integrated with the host operating system's application launcher and does not suffer from a large performance penalty. In order to achieve this goal, we propose to have an isolation framework running on the host operating system and a shared rendering engine for improved performance. Since most of the mobile devices work on the ARM architecture, we plan to evaluate how different virtualization techniques perform on ARM.

Our preliminary investigations evaluate the difference in performance between emulating ARM-on-ARM vs x86-on-ARM and our results show that ARM-on-ARM virtualization has much lower performance overhead. We also detail other virtualization techniques that we plan to investigate which include traditional techniques like User Mode Linux and OS-virtualization tools like Linux Containers(LXC). Linux Containers present a low overhead technique of isolating different virtual environments and we believe that they may represent the ideal balance between security and usability.

We believe that an integrated solution where untrusted applications are integrated with mobile device would enable wider adoption. The rest of this report is structured as follows: Section 2 presents an overview of the design and also describes in detail some of the design choices which we make. Sections 3 describes the preliminary implementation and deployment of virtualization on two smart phones and also discusses details about porting the X server. Evaluation of our existing system is presented in 4. Section 5 discusses existing efforts in this area and section 6 concludes.

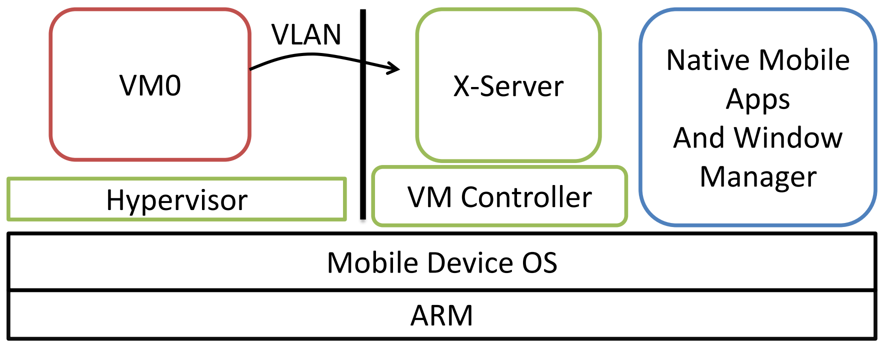


Figure 1: Architecture diagram

2 Design

2.1 Design Overview

Our architecture has two main components: the virtualization framework, and the integration front-end. The virtualization framework contains the hypervisor (KVM, User-Mode Linux, or L4 based) or a container-ready kernel (Linux Containers-LXC) as well as each of the spawned VMs or containers (see Figure 1). A VM would contain a thin OS for running the third-party application and a container could contain one or more applications. Initially we intended to explore both x86-based Operating Systems as well as ARM based systems. However, initial results show that x86-based Operating Systems will be far too slow. Although an x86-based OS would have enabled a more diverse set of target applications, ARM-based systems have less virtualization overhead and some options may not be prohibitively slow.

The other component is the integration front-end. This contains a light X-server that has been integrated into the host OS, and also contains the VM/container controller. The controller will either run inside the X-server as a native application (ARM-based), or as a separate application within the host OS or containers. The controller will be the user interface that controls launching, switching, suspending, resuming, migrating the VMs or containers, as well as ensuring the X-server is running properly.

A shared X-server removes rendering overhead from each VM or container and reduces the complexity in composing the UI. We choose to share the rendering state between third-party applications for performance reasons and simplicity of design, but ensure isolation from native applications. Additionally the X-server runs native code, which allows for it to be less CPU intensive than running inside a VM or container.

We aim to make the virtualization framework device independent, which is important because we anticipate this to be the more complex component. The integration front-end will have to be ported for each new device we support, but we will strive to make its implementation simple. Our first implementation focuses on one particular device and OS and support migration of applications from desktops.

In summary, we leverage an existing virtualization environment to build a secure, usable and portable framework for mobile device virtualization. Our contributions are focused on providing the ability for live migration of applications with deep integration to the mobile device. Our proposed security model is an effective method to isolate applications and we find that our design ideas are similar to other recent efforts [25] in isolating untrusted code.

2.2 Virtualization on ARM

The two primary goals of our design is to a. Provide isolation of selected applications on mobile phones and b. To build a usable solution which can be integrated seamlessly into the host operating system of the smartphone. To achieve the first goal, we first investigated Type-II virtualization techniques which allow a guest operating system to run from within the host operating system. The ARM architecture is not directly virtualizable as there privileged instructions which when executed by the guest operating system, do not trap to the host kernel. There are many existing techniques like dynamic binary translation, trap-and-emulate using hardware extensions, translate to trap, paravirtualization which have been used in existing virtualization solutions for x86. QEMU [21] is one of the more recent, popular and opensource virtual machine monitors that can be used to run operating systems built for different architectures to run on different machines. QEMU relies on dynamic binary translation and has been ported to run on multiple platforms like ARM, PowerPC, i386 etc. Our first implementation uses QEMU and attempts to address the initial design goal of evaluating the overhead of dynamic binary translation on a smartphone running an ARM processor. Results from our preliminary investigations are presented in 4 and we plan to investigate other techniques listed below.

2.3 Other Virtualization Techniques

The following list presents the different techniques that we are considering to have a faster, more usable and secure solution.

2.3.1 KVM

The Kernel Virtual Machine Monitor (KVM) is a virtualization technique in which the Linux kernel plays the role of a hypervisor. In traditional virtualization tools like Xen [20], a hypervisor manages the scheduling, memory management and driver support for the different guest operating systems. Since the linux kernel already performs most of these tasks for the host operating system, it is efficient to re-use this functionality for the guest operating systems too. KVM consists of a kernel module, which introduces a guest mode, page tables and handles privileged instructions through a 'trap-and-emulate' scheme. On the x86 architecture KVM uses hardware virtualization extensions like the Intel VT or AMD-V for the same. The ARM architecture is not strictly virtualizable as there are privileged instructions which do not trap to the kernel and therefore a simple 'trap-and-emulate' approach cannot be used. Hence more complex techniques like dynamic binary translation, translation to add software interrupts or paravirtualization are required for porting KVM to ARM. There have been some initial attempts to port KVM to ARM [18] but the high performance overhead reported leads us to explore other solutions for isolation and security.

2.3.2 User Mode Linux

User Mode Linux (UML) is a virtualization technique in which the guest operating systems run as user mode processes inside the host. When compared to hypervisors like VMWare ESX or Xen, UML offers a simpler solution and is patched into the Linux kernel source tree. The first version of User Mode Linux used *ptrace* to virtualize system calls and modify and divert them into the user space kernel for execution. Later versions of UML introduced the Separate Kernel Address Space (SKAS) mode where the UML kernel runs in a different address space from its processes. This addresses security issues by making the UML kernel inaccessible to the UML processes and also provides a noticeable speedup. Also this technique is only effective when the architecture of the guest operating system is the same as the host and fits the design constraints of our problem. User Mode Linux was built originally on the i386 architecture and has since been extended to the PowerPC and x86_64 architectures. Technically it should be feasible to port UML to ARM architecture.
Some downsides of using UML are that the amount of overhead is relatively large for workloads which have a number of interrupts and the project is not under active development anymore. We plan to explore porting UML to ARM and measure the overhead caused in the weeks ahead.

2.3.3 Linux Containers

Linux Containers (LXC) implement OS-level virtualization techniques in order to run a number of isolated virtual environments on a single host. LXC differs from conventional virtualization techniques, which generally require the installation of guest OSes. The isolated virtual environments or containers, are built upon other Linux security mechanisms. Resource namespaces allow containers to manipulate the accessing of processes, files, and hardware resources. Control groups are used to limit the resources used by a container. Capability bounding sets reduce a container's privileges.

Unlike with conventional virtualization, LXC has almost no overhead as it does not require guest OSes or dynamic binary translation. On the other hand, because it is a OS-level virtualization technique and runs processes on a Linux kernel, other OSes such as Windows and other UNIXes (BSD or OSX) can not be used as containers.

The main obstacle to using Linux Containers are the requirements. LXC requires a Linux kernel version 2.6.27 or greater, and the Pre is running 2.6.24. The kernel functionality would have to be back ported in order to work with the Pre. The Android is running 2.6.29; however, porting the LXC tools to Android will likely be very difficult.

3 Implementation

3.1 Cross-Compilation

Nearly all modern smartphones run ARM processors. On the other hand, nearly all modern desktops run x86 or x86_64 processors. As a result of the architectural differences, binaries created in normal desktop PC environments will not work on smartphones. Additionally, the devices themselves are rather limited with respect to building code. Relatively limited cpu, memory, disk, coupled with input and usability concerns make this an unattractive way to compile code. A number of devices have lacking userspace environments that would make compiling code on them even more difficult. Our solution is to use a cross-compilation environment to use a faster desktop to write our code targeting the various mobile platforms. We build upon the scratchbox2 environment [9], in particular using the cross-compilation enviroment created by webos-internals [14].

Scratchbox2 is a cross-compilation engine that uses a combination of emulation and library interposition to make it easier to cross-compile code bases that don't otherwise provide a means to cross-compile. The result is almost a virtualized environment that automatically invokes cross-compilation tools as required, abstracting away much of difficulties that cross-compilation can usually entail.

Important features in our environment include 1)a system of dependency resolution that allows us to conveniently build and package our applications more easily; 2)staging headers and device libraries such that we can use the same ones that are available on the target device; 3)static or dynamically linked executable compilation. This allows us to conveniently build for multiple platforms from the comfort and reliability of our own systems. Presently we use this to target the Palm Pre as well as android devices, in particular the Motorola G1.

3.2 libc vs bionic

Despite the fact the Pre and the Android are both running Linux OSes on an ARM chip, the environments they provide for our project are enormously different. The Pre is bundled with a large suite of common Unix applications and libraries, while the Android diverges, running minimalistic libraries and lacking many common libraries and applications. Of these differences, the C compiler library is probably the most substantial. The Pre uses an unmodified version of glibc and the Android uses Bionic, a lightweight and small C compiler library. Many Linux applications have glibc as a dependency, and as a result, builds for the Android must have statically link libraries. Between this and the lack of useful applications (gdb and other debugging tools) development on the Android has proven much more difficult.

3.3 X-Server

We made a number of important design decisions, many of which were in response to unexpected implementation difficulties. The result of our X server (including a fixed Xsdl implementation) work is available on the website[1].

3.3.1 DDX

The X-Server architecture contains multiple Device Dependent X (DDX) implementations. The most used one is 'xfree86', but there are others including 'kdrive' [15]. We chose to use kdrive because of the Xsdl component it contains, which allows one to run X using SDL as a backend. Unfortunately Xsdl is so out of date that it was recently removed from the X project altogether due to being broken and unmaintained.

From X version control: "if anyone uses this in production, a big scary monster will eat them" [17]. The result of this was much work spending fixing Xsdl and bringing it up to date to work with the rest of X. Fixes including interactions with the X server, as well as fixing the rendering code and the input handling.

3.3.2 SDL and GLES

As described above, the code we started with used Simple DirectMedia Layer (SDL) [10] for input and rendering. Once we had achieved this functionality, we noticed the display lagged when doing even basic things like moving the cursor. Previous experience working with SDL on this device suggested that using GLESv2 [7] and custom shaders would improve a task even as simple as blitting, so we ported the rendering bits to use GLESv2.

3.3.3 Devices that don't support SDL

Although the Pre has support for SDL, many devices don't, and that's something we've taken into consideration. The kdrive structure can be made rather portable: at it's heart it just needs something that can render a pixelbufer, and feed it input (either event driven or by polling). This means that we could potentially support Android devices through Java Native Interface (JNI) [1] passing the buffer to a java application to blit and the java application blitting to the screen, as well as gathering input and feeding it to the X server.

3.3.4 Keyboard

Keyboard support is very important when using applications. However it was a stumbling block for us for two main reasons: 1) mapping SDL to something the X server can use 2) adding support for keys and features that are not on the original device.

It is common for keyboards, particularly on phones, to have each key have multiple uses when pressed with a special modifier. As an example, on the Pre, 'orange' plus 'q' is the '/' key. This presented a problem because this means capturing the modifier requires creating a state machine to process the input as opposed to a simple lookup table. We ended up using xkb to do this, and the results are in the xkeyboard-config git repository.

The second issue is that many keys that are required for doing something as simple as using 'xterm' simply don't

[1]http://wdtz.org/cs523/

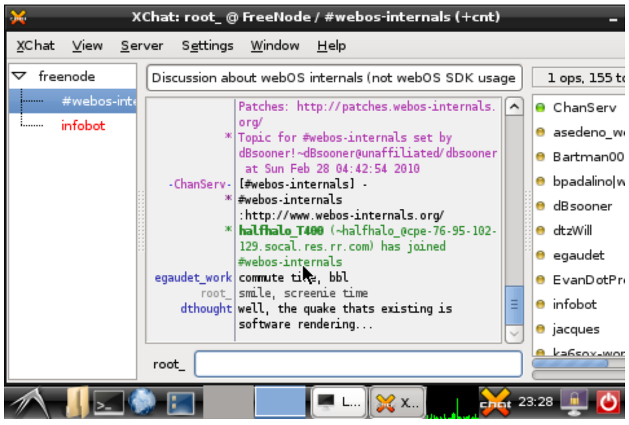
exist on most phones. Examples of such characters include the pipe '|' character, '>', '<', arrow keys, and more. We currently support many such keys on the Pre through more customizations to the xkb layout and hope to support other devices as we add support for them as well.

3.3.5 Future: Integrating even more

A primary goal of our project is to cleanly integrate into the host X server. While our current implementation is a great step and does integrate as a window in the existing windowing system for the device, there is an issue. Presently the X server will render everything into one window (see Figure 2), which requires a window manager to manage the windows. This is bad both because it is hard to use but also because it is a hard break from the goal of integrating with the parent window manager–now the user has to think about it as two separate systems.

One solution to this is to take advantage of the work done on rootless X [8]. "The generic rootless layer allows an X server to be implemented on top of another window server in a cooperative manner. This allows the X11 windows and native windows of the underlying window server to coexist on the same screen. The layer is called "rootless" because the root window of the X server is generally not drawn. Instead, each top-level child of the root window is represented as a separate on-screen window by the underlying window server" [8].

Another idea is to take advantage of standard X notification events [16] and hook into mobile device notification systems, which both WebOS and Android support.



```
                Patches: http://patches.webos-internals.
                org/
              * Topic for #webos-internals set by
                dBsooner!~dBsooner@unaffiliated/dbsooner
                 at Sun Feb 28 04:42:54 2010
    -ChanServ-  [#webos-internals] -
              * #webos-internals
                :http://www.webos-internals.org/
              * halfhalo_T400 (~halfhalo_@cpe-76-95-102-
                129.socal.res.rr.com) has joined
                #webos-internals
  egaudet_work  commute time, bbl
         root_  smile, screenie time
     dthought   well, the quake thats existing is
                software rendering...
```

Figure 2: X server running icem and xchat

4 Evaluation plan

4.1 Preliminary Virtualization Results

Currently, we have evaluated the overhead of virtualization with a QEMU-based implementations running various distributions of Linux on both the Android and the Palm Pre as shown in Table 1. The most successful implementation used ARM on ARM virtualization and a basic ARM kernel image provided along with QEMU. Unfortunately, even this implementation was far too slow, taking 52 seconds to boot on the Pre and 154 seconds to boot on the Android. Additionally response times were terrible, often taking several seconds to display text input. A Debian ARM Lenny image took over 15 minutes to boot on the Palm Pre and crashed on the Android during boot. x86 emulation on ARM was also tested, with a TTY-Linux image; however, this was the slowest by far, taking over half an hour to boot on the Palm Pre. On the Android, x86 guest mode QEMU would not even run. Overall these experiments suggest that binary translation is not usable and that we need to explore for different virtualization techniques listed in Section 2.

4.2 X-Server numbers

An important part of our system is the X server to visualize and interact with the applications. While our current design is somewhat limited in that it has too many layers of abstractions (using SDL as the backend), we've taken efforts to make the server run faster, which resulted in a much better user experience. The biggest performance improvement was moving from basic SDL to SDL-GLESv2 which improved the "feel" of X and the applications inside of it noticeably. To try to capture this speed improvement we ran x11perf, which helps quantify the performance improvements. As shown in Table 2, there was noticeable improvements in a number of tests. These tests we run from a Debian chroot, using localhost communication (not domain sockets) with the server, on the Palm Pre. The tests were arbitrarily selected, with an attempt at finding representative ones. These numbers should only be taken as illustrating the general performance improvements, not as an accurate measure of what real applications will be like.

Benchmark	Xsdl	Xsdl-gles	%Speedup
oddtilerect100	9950	11500	15.57%
scroll100	6680	7700	15.26%
copy100	2940	3440	17.01%
rect100	15700	18900	20.38%
fcircle100	8130	9940	22.26%
ftext	481000	556000	11.43%

Table 2: X server rendering with x11perf

	Palm Pre Cortex-A8 256MB RAM	Android ARM 1136-JS 128MB RAM
Basic ARM kernel	52	154
Debian ARM Lenny	1186	Crashes during boot
TTY-Linux-i486	> 2000	Unable to get x86 guest mode qemu to run on arm

Table 1: Virtualization Results: Kernel Boot time in seconds

4.3 Future Evaluation

Our goal for this project is to integrate an isolation technique which is usable and is integrated with the mobile device. We propose to evaluate the overhead due to the isolation framework and also ensure correctness using standard benchmarks. In particular, we plan to evaluate the latency involved in running an interactive application and measure how much benefit is obtained due to the shared X server running natively. We also plan to profile the memory and cpu usage overhead entailed by our design and compare the same with the application running natively. The VM Controller will be integrated with the window manager of at least one mobile device and we will demonstrate the ability to start and stop applications using the controller.

5 Related Work

Currently, there are many solutions available for virtualization on desktop environments. VMware is a popular closed source solution which implements a variety of virtualization techniques and is used in both industry and academia. KVM [28], QEMU [21], and XEN [20] are all open source solutions, implemented using a variety of virtualization techniques. These solutions cannot be directly used in mobile environments for performance and usability reasons.

Recently there has been a surge of research in the area of mobile virtualization. One such solution is MobiVMM [30], which prioritizes performance and security at the cost of usability and portability. Work has been done to port KVM to ARM [18], focusing on performance and functionality. All of these solutions either dual-boot the OS or require disabling the phone's existing runtime stack.

VMware's MVP project [29] is most similar to ours. They introduce a very thin Type I hypervisor with an emphasis on usability, performance, and security–without sacrificing the phone's functionality. However their implementation does not integrate with the host OS, but rather replaces and contains it. This is useful, but tangential to our work. Open Kernel Lab's OKL4 [26] is another implementation of a thin Type I hypervisor and is very similar to MVP. As described in Section 2.1, we aim to provide a Type II Virtual Machine (VM) and prioritize live migration capabilities as well as usability. Instead of virtualizing the existing OS to protect other Virtual Machines, we assume it to be trusted and only isolate third party applications. This provides for a very different architecture in our implementation.

There also is ARM's TrustZone [19] which is aimed at creating a secure "TrustZone", primarily for use in DRM, bank transactions and other similar setups. The goal is to protect a specialized app from the rest of the system (and protect, for example, secrets and keys from leaking out of this zone). We aim to to do the opposite: we trust the host OS and are protecting it from the guest applications.

6 Conclusion

Initially, we had planned to focus on security and portability of our implementation. However, as we progressed, we discovered that performance could not simply be ignored. We discovered that QEMU was unusably slow in a smartphone environment and we had to step back and reconsider our goals in respect to performance.

We experienced a lot of success in porting X to the Pre, and although the preliminary virtualization attempts were unusably slow, the remaining options are hardly exhausted. In terms of performance Linux Containers (LXC) looks very promising. Despite needing to backport the needed kernel functionality to support LXC on the Pre, we remain optimistic and will continue working towards our goal of a usable, fast, and secure solution for running native and untrusted apps on smartphones.

References

[1] `http://java.sun.com/j2se/1.4.2/docs/guide/jni/index.html`.

[2] 16 percent of workloads are running Virtual Machines. `http://www.gartner.com/it/page.jsp?id=1211813`.

[3] Android Security and Permissions. `http://developer.android.com/guide/topics/security/security.html`.

[4] Cyber-criminals target mobile banking. `http://www.v3.co.uk/vnunet/news/2173161/cyber-criminals-target-mobile`.

[5] iPhone Privacy. http://seriot.ch/resources/talks_papers/iPhonePrivacy.pdf.

[6] Mobile Phones, The Next Frontier. http://blogs.vmware.com/console/2009/08/mobile-phones-the-next-frontier.html.

[7] Open GL Embeded Systems. http://www.khronos.org/opengles/2_X/.

[8] Rootless X. http://cgit.freedesktop.org/xorg/xserver/tree/miext/rootless/README.txt.

[9] Scratchbox 2. http://www.freedesktop.org/wiki/Software/sbox2.

[10] Simple DirectMedia Layer. http://www.libsdl.org/.

[11] Vulnerability Summary for CVE-2009-0475. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-0475.

[12] Vulnerability Summary for CVE-2009-2204. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-2204.

[13] Vulnerability Summary for CVE-2009-2692. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-2692.

[14] WebOS Internals. http://www.webos-internals.org.

[15] X Glossary. http://www.x.org/wiki/Development/Documentation/Glossary.

[16] X Notifications. http://www.galago-project.org/specs/notification/0.9/index.html.

[17] X Version Control. http://cgit.freedesktop.org/xorg/xserver/commit/?id=52bc6d944946e66ea2cc685feaeea40bb496ea83.

[18] D. A. Andreas Nilsson, Christoffer Dall. Android Virtualization. http://www.chazy.dk/android-report.pdf, 2009.

[19] ARM Ltd. ARM - TrustZone. http://www.arm.com/products/processors/technologies/trustzone.php.

[20] 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, 2003.

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

[22] A. Bose and K. Shin. On Mobile Viruses Exploiting Messaging and Bluetooth Services. *Securecomm and Workshops, 2006*, pages 1–10, 2006.

[23] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, page 286. USENIX Association, 2005.

[24] L. Cox and P. Chen. Pocket Hypervisors: Opportunities and Challenges. In *Eighth IEEE Workshop on Mobile Computing Systems and Applications, 2007. HotMobile 2007*, pages 46–50, 2007.

[25] C. Grier, S. Tang, and S. King. Secure web browsing with the OP web browser. In *Proceedings of the 2008 IEEE Symposium on Securiy and Privacy*, 2008.

[26] O. K. Labs. OKL4 Microvisor. http://www.ok-labs.com/products/okl4-microvisor.

[27] Pandora Research. Vodafone distributes Mariposa botnet. http://research.pandasecurity.com/vodafone-distributes-mariposa.

[28] Qumranet. Kernel-Based Virtual Machine. [Online], 2009. Available: http://linux-kvm.org.

[29] VMware. VMware MVP (Mobile Virtualization Platform). http://www.vmware.com/products/mobile.

[30] S. Yoo, Y. Liu, C.-H. Hong, C. Yoo, and Y. Zhang. MobiVMM: a virtual machine monitor for mobile phones. In *MobiVirt '08: Proceedings of the First Workshop on Virtualization in Mobile Computing*, pages 1–5, New York, NY, USA, 2008. ACM.