MoZyg: Secure Framework for Cross Platform Applications on Mobile Devices

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

Abstract

Modern day smartphones are plagued with security vulnerabilities. As smartphone adoption continues to increase, and more sensitive information is stored on them, there is an increasing need for a more secure solution. We developed *MoZyg*, a system for mobile devices that uses lightweight OS-level virtualization to contain and isolate applications in order to increase security on these devices. Our system has negligible overhead (5-10%), while providing strong security guarantees for the applications within. *MoZyg* is designed to be added to an existing mobile device to provide security guarantees for native applications while maintaining the original device experience. Additionally, *MoZyg* allows the use of desktop Linux applications on smartphones (while guaranteeing a secure execution context) enabling a much wider variety of applications to be run on mobile devices than would previously be possible.

1 Introduction

Today, 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 [25, 3, 4] 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 [2] provide process level isolation among applications, operating system bugs such as [16, 15, 17] allow malicious applications to take over the device. Recent reports have even found that some smart phones had the Mariposa botnet pre-installed [30]. Virtualization can be useful for secure isolation of third party code from confidential data and provides greater defense-in-depth against attacks on the system.

In recent years, virtual machines have become prevalent in cluster computing environments [1] 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 [26] and industry [9] about the benefits of virtualization on these devices. Virtualization provides better security guarantees in mobile devices than current solutions offer.

Existing solutions for isolation of mobile applications introduce a Type I hypervisor [32, 28] which manages the phone's hardware and allows running multiple operating systems on top. Popular Type II hypervisors like KVM and QEMU either suffer from a large performance overhead (refer Section 6) or are otherwise unusable. We propose to have isolated applications which are integrated with the host operating system's application launcher and do not suffer from a large performance penalty. In order to achieve this goal, we have an isolation framework running on the host operating system and a shared rendering engine for improved performance.

We adopt an OS-level virtualization tool called Linux Containers (LXC). Linux Containers present a low overhead technique of isolating a process within a container. A container is made up of several namespace virtualization techniques, where the UID, user namespace, network resources etc. are virtualized in order to prevent the process inside the container from interfacing external data or resources. Our tests show that the performance overhead is within 5-10%, proving to be a promising balance between isolation and performance.

The rest of this report is structured as follows: Section 2 presents an overview of the design and the next section describes in detail some of the design choices we make. Sections 4 describes our implementation and deployment of *MoZyg* on smartphones. Evaluation of our system is presented in Section 5. Section 7 discusses existing efforts in this area and section 8 discusses future work.

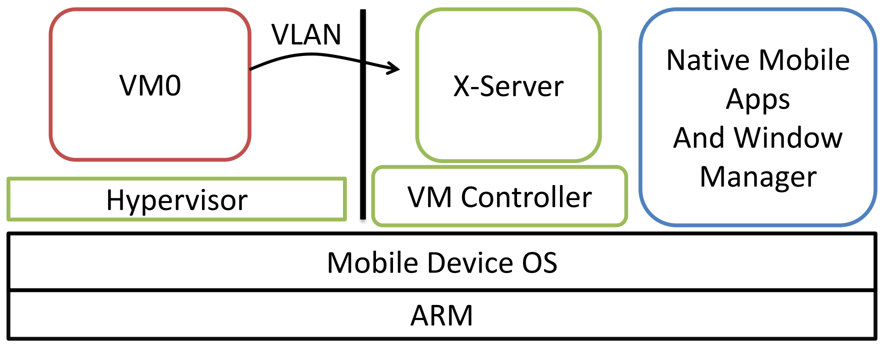


Figure 1: Architecture diagram

2 Design Overview

Our architecture has two main components: the virtualization framework, and the integration front-end. The virtualization framework contains a container-ready kernel as well as each of the spawned containers (see Figure 1). A container could contain one or more applications. Our initial design goals were to explore virtualization of x86-based operating systems as well as ARM based systems. However, our results 6 show that these solutions will be far too slow and hence we choose to use OS-level virtualization techniques.

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 container controller. The controller runs inside the host OS. The controller will be the user interface that controls launching, switching, suspending, resuming, migrating the containers, as well as ensuring the X-server is running properly.

A shared X-server removes rendering overhead from each 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.

In *MoZyg*, the virtualization framework is designed to be device independent. The integration front-end will have to be ported for each new device we support, but its implementation is relatively simple. Our first implementation focuses on one particular device (Palm Pre) and OS (WebOS).

In summary, we leverage the existing linux containers architecture to build a secure, usable and portable framework for mobile device virtualization. Our contributions are focused on providing the ability for secure execution of applications while also providing 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 [27] in isolating untrusted code.

3 Design

3.1 Secure Isolation

The two primary goals of our design are: to provide isolation of selected applications on mobile phones; to build a usable solution which can be integrated seamlessly into the host operating system of the smartphone. To achieve these goals we either need a type-II hypervisor which can run within the host operating system or employ OS-level virtualization techniques like containers. Unlike conventional virtualization, containers have almost no overhead as they do not run the entire software stack of the guest operating system. On the other hand, because the virtualization is facilitated by the operating system, other OSes or architectures cannot be used within a container. In our design, we propose to use operating system level virtualization techniques of which Linux Containers and OpenVZ [11] are popular implementations.

3.2 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 [33]. We list the different resources isolated using containers and the techniques used for them below.

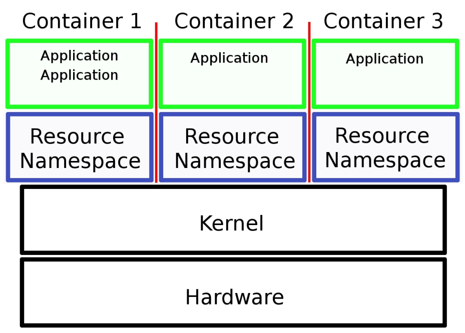


Figure 2: Linux Container Architecture

3.3 Host Identification

On Linux machines, the identity of a system is usually established through the command ‘uname’. This command reads information stored in the structure ‘utsname’. The structure includes the name of the operating system in use and the current version of the operating system. The structure also contains the hostname of the machine and this is used to identify the machine in network communications. Linux Containers implement a per-process utsname namespace and this enables each container to have a different hostname.

3.4 Process Identifiers

Process identifiers (PIDs) are isolated using PID namespaces such that with respect to each container the set of PIDs appears like a standalone machine. This also allows two processes to have the same PID in different containers. Processes can be launched in a new PID namespace using the clone or the unshare system-calls. This is an important feature for enabling migration of containers between hosts as the PID values can be the same at the source and destination. Also, PID namespaces are hierarchical; tasks in the original namespace can see the PIDs of tasks in the new namespace but not vice-versa.

3.5 Inter-Process Communications

Inter-Process Communication (IPC) between processes uses shared objects like shared memory segments, message queues etc. To create isolated environments, it is necessary that processes from one container cannot share data or communicate with other containers. This is achieved by introducing IPC namespaces which creates separate set of IPC objects for each container.

3.6 User Namespaces

User namespaces supplies additional UID tables in order to allow a UID to be namespace specific. With user namespaces, different users can be associated with the same UID across different processes.

3.7 Network Devices

The network namespaces assign a private set of network resources, including IP addresses, routes, sockets to any number of processes. This allows for shared names among different namespaces and isolation between those namespaces. This provides substantial improvements in security and network resource management. Network namespaces are implemented using virtual ethernet devices and containers can bind to the same ports without interfering with each other. This is a useful feature as, for example, we could have two webservers listen on port 80 in different containers.

3.8 Readonly-bind mounts

Bind mounts provide the ability to remount part of a file hierarchy at a different location while it is still available at the original location. This allows for read-only accesses to a filesystem mounted read-write. It guarantees that one process can read and write to the filesystem, while others can only read from it, regardless of their privileges.

3.8.1 Copy-on-Write filesystems

One of the techniques used to prevent duplication of system directories such as /bin, /lib across containers is to mount them as read-only bind mounts from native system into a container. However this means that processes running within containers would be restricted and not able to perform tasks like installing new binaries into /bin. In order to enable each container to make modifications to the system directories and avoid making multiple copies, we use Unionfs [34] a file-level copy-on-write filesystem.

Unionfs is a file system which aims to maintain UNIX semantics while providing advanced namespace-unification capabilities. It allows read-only and read-write branches to be inserted into the same tree and provides support for snapshots. Unionfs is often used in scenarios where a base OS image is provided on a read-only CD-ROM and any changes made the user are stored in a separate read-write directory. Our use-case is similar as the phone’s native OS image is mounted read-only and any changes made by each container is stored in its own separate read-writeable directory.

3.9 Other features

In addition to the above mentioned isolation features, the Linux Container project adds support for *Control Groups* which can be used to limit the resources used by a container. Additionally, capability bounds can be used to restrict the privileges of a container and these features help ensure that the resources in a machine can be isolated and fairly shared between different containers.

4 Implementation

We implement *MoZyg* on Palm Pre which run the WebOS operating system based on Linux kernel version 2.6.24.

4.1 LXC and module compatibility

While experimenting with new kernels on WebOS devices, we encountered some issues that required work arounds. Like many mobile devices the Palm Pre ships with a number of binary modules for which the source code is unavailable. For example, the wifi drivers "sd8xxx.ko" and "uap8xxx.ko" that ship with the device are not open-source and so we only have access to the final binary module. This is not uncommon for linux drivers, but presents us with two issues, both having to do with maintaining the kernel application binary interface (ABI).

The first issue was regarding the 'MODVERSIONS' config option in the kernel [6]. This config option helps identify symbols by appending a CRC of their contents to the symbol name. This was an issue because our changes slightly modified the definitions of some symbols, and this resulted in mismatched version information. Our solution was to build the kernel without support for this, but this results in lesser safety guarantees when loading kernel modules. This is not a security concern since loading modules is a privileged operation anyway, but is undesirable as the safety checks help prevent the user from loading incompatible modules.

The second issue was that the ABI of our modified kernel needed to match that of the stock kernel, thus restricting the kind of changes we could make in the kernel. The Palm Pre ships with a modified version of linux 2.6.24 [12], but full support for LXC was not merged into the linux mainline until 2.6.26 [7]. In 2.6.24 there is partial LXC support, and we had to backport some of the other components. However we were unable to get full LXC support since some features (such as network namespaces) rely on significant changes in the kernel architecture. However, this is not a fault in our design and a device manufacturer most likely would not be faced with this issue, being in the position to acquire or build the related modules for the kernel version they desired.

4.2 X-server

The X-server serves as the graphical framework that integrates the applications into the host environment. As part of our efforts to bring X11 to the phones, we had to make a few decisions that are detailed below.

4.2.1 DDX

The X-server architecture contains multiple Device Dependent X (DDX) implementations. The most used one is 'xfree86', but there are others, such as 'kdrive' [18]. We chose to use kdrive because of the Xsdl functionality it contains, which allows one to run X-server using SDL as a backend. Unfortunately Xsdl is so out of date that it was recently removed from the X-server project altogether due to being broken and unmaintained.

From X-server version control: "if anyone uses this in production, a big scary monster will eat them" [20]. The unfortunate result of this was much work spent fixing Xsdl and bringing it up to date, in order to work with the rest of X-server. Fixes included interactions with the X-server, as well as fixing the rendering code and the input handling.

4.2.2 SDL and GLES

As described above, the Xsdl kdrive code we started with uses Simple DirectMedia Layer (SDL) [14] 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 [10] and custom shaders would improve a task even as simple as blitting (graphical copy), so we ported the rendering bits to use GLESv2. We provide numbers for the resulting increase in performance and discussion in Section 5.2.

4.2.3 Devices that do not support SDL

Although the Palm Pre has support for SDL, many devices do not, and that is something we have taken into consideration. The kdrive structure can be made rather portable: at its 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) [5], passing the buffer to a java application to blit, which gathers input and feeds it to the X-server.

4.2.4 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 used X-Keyboard-Config (xkb) for this.

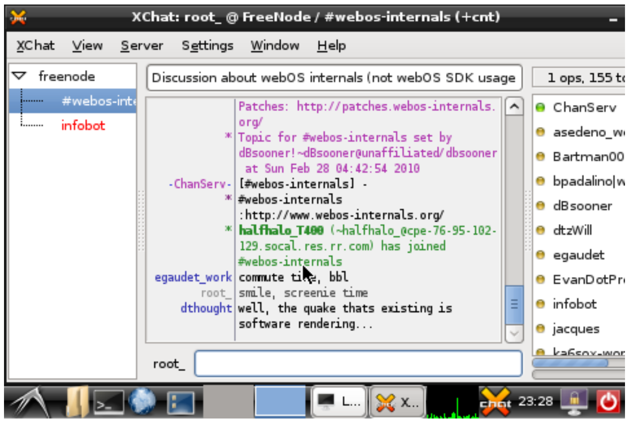
The second issue is that many keys that are required for doing something such as using 'xterm' simply do not exist on most phones. Examples of such characters include the pipe '|' character, '>', '<', arrow keys. We currently support many such keys on the Palm Pre through more customizations to the xkb layout and hope to support other devices.

4.2.5 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 3), 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-server [13]. "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" [13].

Another idea is to take advantage of standard X-server notification events [19] 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 3: X-server running icem and xchat

5 Evaluation

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 1: X server rendering with x11perf

5.1 Testing methodology

We perform our experiments on the Palm Pre which has 256MB RAM and runs on a ARM Cortex-A8 processor. All numbers we report were run at least 3 times and averaged. Unless explicitly mentioned otherwise, the variation was nominal.

5.2 X-server numbers

An important part of our system is the X server used 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 have 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 1, 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.

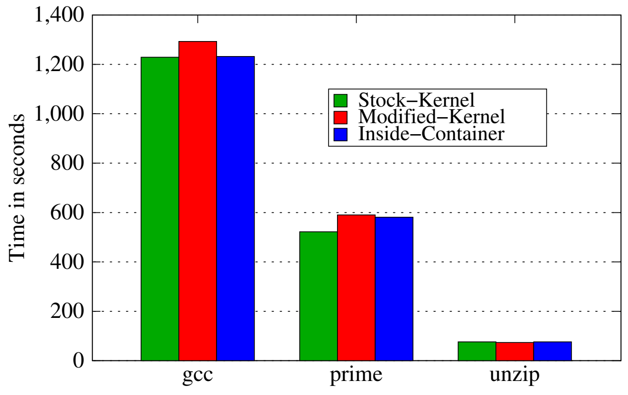


Figure 4: Running time comparison

5.3 LXC Performance numbers

In this section, we measure the overhead of our modifications to the kernel and the overhead due to running processes inside a container. Figure 4 plots the running time of the following applications:

- Compiling apache server version 2.2.15
- Finding prime numbers between 0 and 500,000
- Unzipping a 206.9MB file.

From the figure, we can see that the overhead is approximately within 5%-10% and that running a process within a container does not significantly increase the running time.

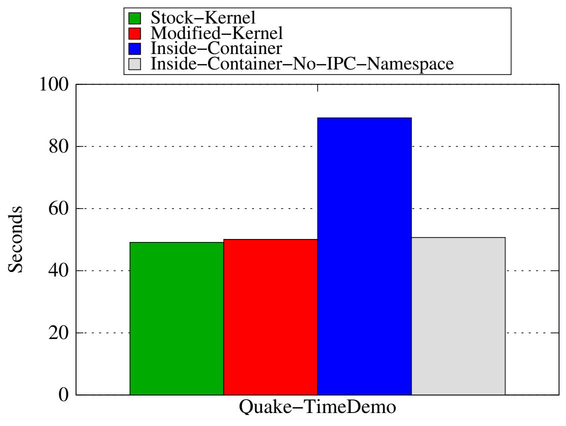


Figure 5: Execution Time for the Quake time-demo

5.4 Quake Time-Demo

To measure the overhead of communicating from the container to an X-server running natively, we run the Quake timedemo. The quake binaries were obtained from a Palm Pre port of the libsdl.org implementation and the command used to run the experiments was 'quake +timedemo demo2'. Figure 5 presents the running time of the demo in different scenarios. From the graph, we can see that the running time is similar for the stock kernel and the modified kernel. When the experiment is run from within the container, we observe that the running time increases by about 60%. We attribute this to the fact that the containers are configured to create a new IPC namespace and hence the application communicates with the X-Server through the loopback interface. When the containers are launched without IPC namespaces, we find that the performance is similar to that of the stock kernel.

5.5 LMbench

Our earlier experiments showed that there was no significant increase in the end-to-end running time of applications running inside a container. We further run micro-benchmarks on the system in order to determine if there is any overhead pertaining to process creation, context switching, or creating files.

LMbench contains a suite of simple, portable benchmarks and is useful for comparing the performance of various UNIX systems. It includes a variety of bandwidth benchmarks, of which the most popular are: cached file reads, memory operations (copy, read, and write), pipe, and TCP. Additionally, it supports a wide variety of latency benchmarks, including: context switching, network connection establishment (pipe, TCP, UDP, RPC), file system operations (creates and deletes), process creation, handling of signals, system call overhead, and memory read latencies. LMbench also supports a variety of multiprocessor tests and includes large databases of results to compare one's results to [29]. LMbench3 is the most mature and commonly used benchmark among the LMbench family. It runs a wide variety of tests, ranging from intensive latency testing of cache misses to extensive context switching testing. LMbench has been heavily used in industry as well as academia [8].

Figures 6 and 7 present results from running LMbench3 on a modified kernel and from within a container, normalized to the time taken on a stock kernel. From the results, we can see that the overhead for system calls and for process creation are negligible compared to the stock kernel. We do not attribute any reason to read and write calls being faster in the modified kernel and believe this to be due to external noise.

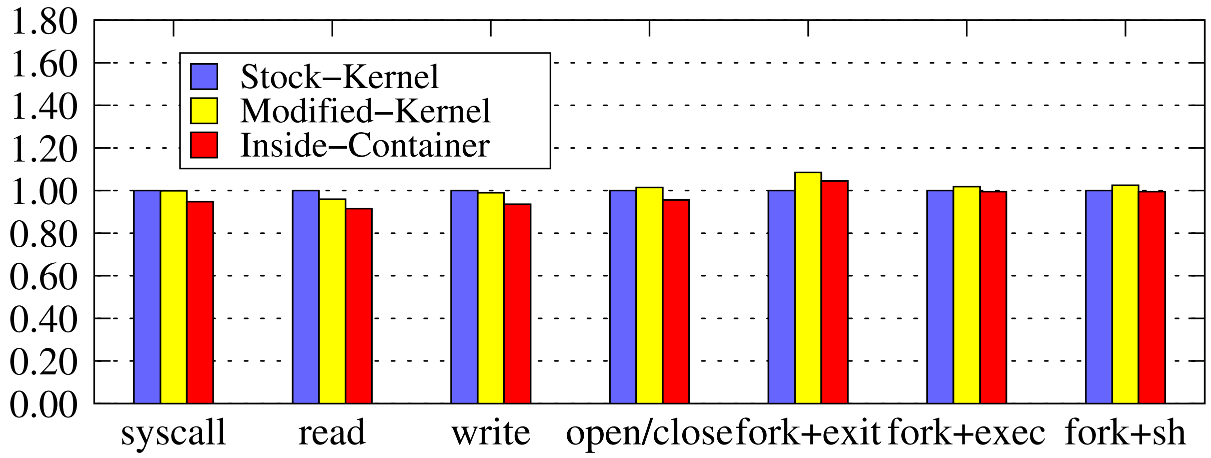


Figure 6: Normalized results from LMbench

6 Discussion

A large component of *MoZyg* is the use of virtualization to achieve our security requirements. Unfortunately, the ARM architecture is not directly virtualizable as there are 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, and paravirtualization which have been used in existing virtualization solutions. We discuss some of these solutions below and why we believe containers are more suitable for smartphones.

6.1 QEMU on ARM

QEMU [24] is one of the more recent, popular and open source 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 used QEMU and attempts to address the initial design goal of evaluating the overhead of dynamic binary translation on a smartphone running an ARM processor. However, we found that the overhead due to dynamic binary translation is quite large without hardware assistance and explore other solutions for providing isolation. We present results from our experiments later in Section 6.4.

6.2 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 [23], 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 [21] but the high performance overhead reported led us to explore other solutions for isolation and security.

6.3 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 hence this fits the design constraints of our problem. User Mode Linux was

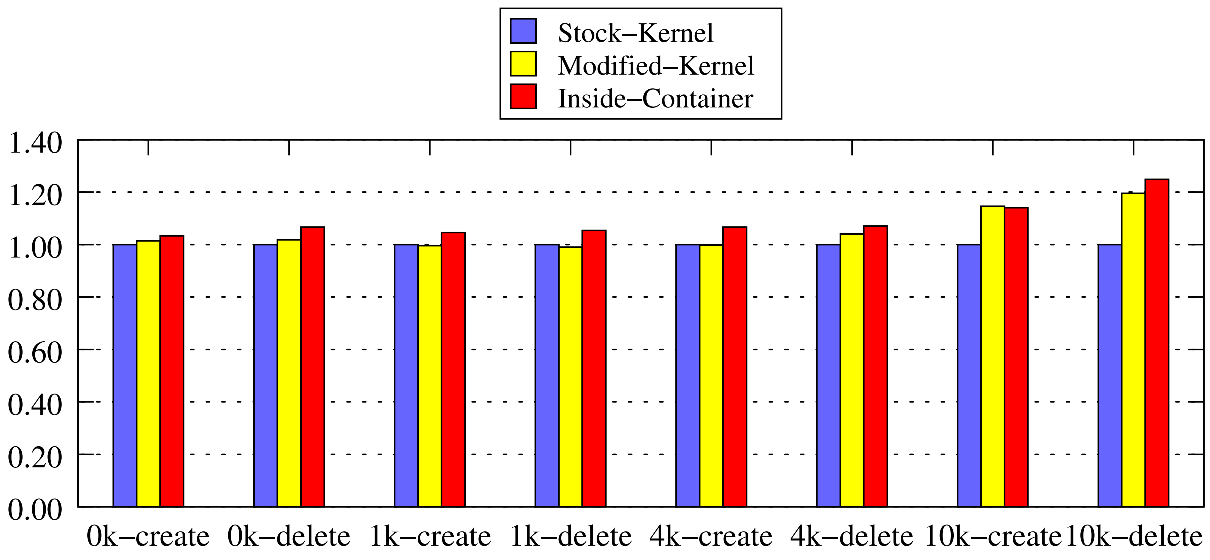


Figure 7: Normalized results from File System create/delete benchmarks in LMbench

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.

6.4 Why not QEMU

Given the above results, the only system we were able to evaluate as a solution was QEMU (since the other two were unavailable for our platform). We evaluated the overhead of virtualization with a QEMU-based implementations running various distributions of Linux on both the Android and the Palm Pre, and the results as given in Table 2. 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 high, 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 so we explored different virtualization techniques.

7 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 [31], QEMU [24], and XEN [23] 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 [35], which prioritizes performance and security at the cost of usability and portability. Their work has several innovations, using ARM-specific hardware to assist in virtualization. However, their numbers demonstrate a noticable overhead above native execution and demonstrate only the latency overhead of their VMs, not the startup times or the overheads imposed.

There have been efforts to port Xen to the ARM hardware [23]. It supports many operations, including secure booting and storage, access controls, booting of multiple OSes, and static memory partitioning. Unfortunately, the project is still very immature and many Xen tools and features are absent and newer versions of ARM are unsupported. Additionally, it seems little to no work has been done since 2008.

Work has been done to port KVM to ARM [21], focusing on performance and functionality. The paper discusses ARM virtualization and covers how it can use dynamic

	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 2: Virtualization Results: Kernel Boot time in seconds in QEMU

binary translation, translate to trap, and basic block breakpoints to overcome the unvirtualizable hardware of an ARM processor. They discuss the (significant) portions of KVM, which are hardware (x86) specific, and will need to be modified to support ARM processors. While promising, the project still only supports a small subset of ARM processors and the project as a whole is still quite immature.

VMware's MVP project [32] introduces a very thin Type I hypervisor with an emphasis on usability, and security. However their implementation does not integrate with the host OS, but rather replaces and contains it, with the intention of running multiple OSes. The goal would be able to run a work OS side by side with a personal OS, isolating secure data from personal data and vice-versa. While this is useful, it is tangential to our work. Additionally, MVP is still under heavy development, and release has been bumped back to at least 2012.

Open Kernel Lab's OKL4 [28] is another implementation of a thin Type I hypervisor and is very similar to MVP. OKL4 has a very small memory footprint and adds minimal overhead to execution times. It is built upon a lightweight L4 microkernel, and rather than for running multiple OSes, it is designed to run under existing phone and embedded OSes to provide additional isolation, security, and information flow control.

There also is ARM's TrustZone [22] 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, secure data, and other applications from the given guest applications.

8 Future Work

MoZyg already makes good progress towards making mobile devices secure, and bringing desktop applications into that secure context. However, we have a few limitations that we hope to address in future work which we describe below.

8.1 Live Migration

Since we already have the applications running in virtualized containers, an exciting next step would be to support live migration. Linux Containers already supports the pausing and resuming of containers, and future would could build upon this to allow live migration of applications. Given our implementation choice to use the fairly universal X11, one could use such live migration to move applications to/from desktops or other phones. There are of course many implementation details to make this practical, but we believe this could be a useful and exciting direction to take our work.

8.2 Additional Containers Support

As the Palm Pre's stock kernel version is 2.6.24, most of the Linux containers functionality had to be back ported. Unfortunately, a few of the components (such as network namespaces) weren't practical to backport beacuse they rely on important architectural changes between 2.6.24 and 2.6.26. In future work we could finish backporting these remaining components, or port Palm's kernel changes to the 2.6.26 or later kernel to get full LXC support.

8.3 LXC Design

Even after full LXC support, a number of important decisions can explored to see their effect on performance, and evaluate their effectiveness regarding security. In Section 5.4 we demonstrated the effects of enabling or disabling IPC namespaces had on a benchmark that used X-server which ran outside the container. We ran it outside to enforce the explicit communication channels to be over the explicit X-server protocol, but a decision could be made to trade performance for that security enhancement. Other such potential trade-offs include the network topology of the containers and what resources make sense to expose to each container.

9 Conclusion

Today's smartphones can be insecure, and often attempt to curb that by limiting the manner of applications that can be executed. We built and evaluated *MoZyg* to address both these issues, providing a framework to allow the execution of desktop linux applications on mobile devices, as well as providing a safe execution environment for both the new apps and any native application the device wants to run. Our evaluation shows that *MoZyg* does not sacrifice performance while achieving this, and we demonstrate our success in bringing X-server applications to mobile devices. Most importantly, *MoZyg* was built to *add* to the existing device experience, not alter or supplant it, making it suitable and desirable to extend to many mobile platforms.

Finally all of our work (source, modifications, build environments, packaging) are available at the following URL: `http://wdtz.org/cs523`.

References

[1] 16 percent of workloads are running Virtual Machines. `http://www.gartner.com/it/page.jsp?id=1211813`.

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

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

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

[5] Java Native Interface. `http://java.sun.com/j2se/1.4.2/docs/guide/jni/index.html`.

[6] Kernel config options. `http://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt`.

[7] Linux Container Components. `http://lxc.sourceforge.net/`.

[8] LMbench. `http://www.bitmover.com/lmbench/why_lmbench.html`.

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

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

[11] OpenVZ. `http://wiki.openvz.org/Main_Page`.

[12] Palm Pre Open Source Packages. `http://opensource.palm.com/1.4.1.1/index.html`.

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

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

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

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

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

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

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

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

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

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

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

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

[29] L. McVoy and C. Staelin. LMbench: Portable tools for performance analysis. `http://lmbench.sourceforge.net/lmbench-usenix.pdf`, 1996.

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

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

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

[33] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. `http://www.usenix.org/event/sec02/full_papers/wright/wright_html/index.html`, 2002.

[34] C. P. Wright, J. Dave, P. Gupta, H. Krishnan, D. P. Quigley, E. Zadok, and M. N. Zubair. Versatility and unix semantics in namespace unification. *ACM Transactions on Storage (TOS)*, 2(1):1–32, February 2006.

[35] 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.