Trusted Virtual Platforms: A Key Enabler for Converged Client Devices

Chris I Dalton, David Plaquin, Wolfgang Weidner, Dirk Kuhlmann,
Boris Balacheff, Richard Brown

HP Laboratories, Filton Road, Bristol BS34 8QZ
+44-117-3129750

{firstname.lastname}@hp.com

ABSTRACT

This paper introduces our work around combining machine virtualization technology with Trusted Computing Group technology. We first describe our architecture for reducing and containing the privileged code of the Xen Hypervisor. Secondly we describe our Trusted Virtual Platform architecture. This is aimed at supporting the strong enforcement of integrity and security policy controls over a virtual entity where a virtual entity can be either a full guest operating system or virtual appliance running on a virtualized platform. The architecture includes a virtualization-specific integrity measurement and reporting framework. This is designed to reflect all the dependencies of the virtual environment of a guest operating system. The work is a core enabling component of our research around converged devices – client platforms such as notebooks or desktop PCs that can safely host multiple virtual operating systems and virtual appliances concurrently and report accurately on the trustworthiness of the individually executing entities.

Keywords

Trusted Virtualization, Open Trusted Computing, TPM, TCG

1. INTRODUCTION

Machine virtualization allows a single physical machine to run several (perhaps different) operating systems concurrently; it creates the illusion of multiple machines each running its own operating system. Machine virtualization technology has seen significant industry momentum and uptake in areas such as server consolidation, utility computing, and dynamic provisioning of test and development environments.

The industry momentum behind machine virtualization has also started to create exciting opportunities for enhanced secure system and infrastructure design. A traditional weakness with current computing platforms that run an operating system directly above the machine bare metal hardware is that we have no choice other than to trust that operating system as the security gatekeeper on the platform[1].

Architecturally, machine virtualization places a component called a *Virtual Machine Monitor* (VMM) or *Hypervisor* between the OS and the bare machine hardware. The VMM is used to carve up a physical machine into multiple *virtual machines*, each running its own operating system[2]. The VMM gives us a new point where we can introduce OS-independent security controls. In principal, the VMM can be a fairly small and tight code base with much less of an attack surface than a typical OS, making the VMM a more reasonable placeholder for our trust. In practice though, the most popular implementations of VMM layers have, to date, focused primarily on functionality and usability rather than security and trustworthiness.

The first part of this paper looks at our attempt to redress that balance. In particular we look at our efforts around reducing and re-structuring the privileged control plane of the Xen Hypervisor [1], where we describe an architectural approach for splitting up the control plane of Xen into a number of contained and partitioned services sitting above a core management interface. We focus on the control plane because that aspect currently contains the majority of the privileged code forming the Xen Hypervisor. Minimizing and containing the privileged code is important since this is what gives us confidence in the ability of the VMM to uphold its isolation guarantees for guest operating systems. As an example of a partitioned service using our architecture we describe a virtualized GPU and Secure GUI service. This service sits outside the core privileged management interface and offers guest operating systems access to underlying hardware acceleration and 3D features for graphics and a user interface for securely managing interactions with virtual machines on the system.

Our general philosophy on secure systems design revolves around two key points. First, reducing the amount and complexity of code that you have to trust to uphold your security properties or guarantees is a good idea. Our work around reducing and restructuring the Xen control plane fits into this category. Second, anchoring your trust and security properties in hardware is a good idea too. In the second part of the paper we focus on that aspect where we look at combining machine virtualization technology with Trusted Computing Group technology [7][10].

The Trusted Computing Group is a cross-IT-industry consortium formed in 2003 by founding members that include AMD, HP, IBM, Intel and Microsoft. The main output of the group so far is

[1] With the increasing size and complexity of commodity operating systems this is probably not a good idea. McConnell [4] reports an industry average of about 15–50 errors per 1000 lines of delivered code, although not all errors will be sufficiently exploitable to allow security control bypass. Microsoft Vista is estimated to have around 50 million lines of code compared to the 40 million of the earlier Windows XP and 11–12 million of the earlier NT 4.0.

[2] This is commonly referred to as a *guest operating system* or simply a *guest*.

the Trusted Platform Module (TPM)[3] specification. The TPM is a small hardware chip that is bound to the motherboard of a computing platform. It provides a starting point for gaining trust in that platform in a cryptographically strong fashion, and is designed to be immune from software attacks on the platform. The TPM chip offers three main functionalities: It provides cryptographic identities for a platform; it provides mechanisms for creating a protective store on the platform for sensitive data such as encryption keys; and finally, it provides a cryptographically verifiable report on integrity measurements of the software components running on a platform.

When used with a conventional non-virtualized platform the TPM can, amongst other things, be used to ensure that a platform boots with an OS version and configuration that is considered trustworthy[4]. Essentially, the TPM contains a number of protected Platform Configuration Registers or PCRs. The PCRs are used to record cryptographic hashes (known as *integrity measurements*) of the platform software components such as device expansion ROMs and the operating system boot loader as they are loaded and begin to execute. The hashes or *measurements* recorded in the TPM can be compared against policies also held in the TPM that dictate the conditions under which sensitive data such as encryption keys protected using the protective store features of the TPM can be released. The Microsoft Bitlocker[6] full-disk encryption solution makes use of this aspect of the TPM to ensure that the encryption key for the hard disk is released only if the measurements of various aspects of the system configuration such as the boot loader and OS version match the stored TPM policy. This helps to ensure that an attacker can't bypass OS security controls and gain access to the platform hard disk data by booting a platform via removable media such as a CD or USB stick containing an alternative OS.

The TPM specification sets out a base TPM integrity measurement architecture. This prescribes a way of building up a *chain of trust* in the platform so that when interacting with a particular application on a platform, a report can be obtained on the software components that were executed on the platform in order to get that application up and running[5], and upon which that application depends. This report is a list of PCR configuration values cryptographically signed and certified in such a way that it can be verified that the measurements were obtained by a genuine TPM sitting on a physical platform that meets the requirements of the TCG specifications. Remote parties, for example an online banking service, can use this report to establish the trustworthiness of a particular user's client platform[6] before allowing it access to a service.

Our work aimed at using the TPM in combination with machine virtualization technology takes advantage of the base TPM integrity measurement architecture, but it also extends it to make it more appropriate for virtualized environments. As a starting point we use the conventional TPM features to ensure that we can establish whether a physical platform has booted with a virtualization layer that can be trusted – for example, Xen with our reduced and restructured control plane in an approved configuration.

An operating system running within a virtual machine environment depends not only upon the integrity and correctness of the virtual machine hardware interface provided to it by the VMM, but also on the integrity of the various components that sit behind that interface – for example, the components providing emulated device models[7] for that particular virtual machine. The extended integrity measurement architecture that we describe in the second part of the paper allows us to capture all of these dependencies in a convenient and manageable fashion to describe what we call a *Trusted Virtual Platform*.

A Trusted Virtual Platform groups together all the components that form a virtual machine environment and upon which a virtualized OS or appliance depends. As part of its implementation the Trusted Virtual Platform includes a virtual TPM for use by a guest OS. This allows the trustworthiness of a particular virtual entity (guest OS or virtual appliance) to be reported independently from other virtual entities running on the same platform.

Our work here differs from existing work [2] on providing virtual TPMs for guest operating systems in that we have extended the existing TPM measurement architecture so that we can more accurately capture all of the dependencies of a virtual environment. We believe that the traditional TPM measurement architecture with its strictly hierarchical nature cannot capture all the dependencies of virtualized systems with enough detail to accurately reflect and report on the trustworthiness of a particular virtualized instance.

Additionally, the Trusted Virtual Platform concept allows the management of integrity and security controls for a particular guest OS to be abstracted away from the overall virtualization layer management software and hence more easily and reliably managed. For example, the definition of a Trusted Virtual Platform configuration for a particular guest OS can include a set of firewall rules that should be applied to that virtualized OS irrespective of whether that virtualized OS itself hosts its own firewalling capability. Components within the Trusted Virtual Platform interpose on all security-related lifecycle actions for a virtual machine hosting a particular guest OS and act as an enforcement point for the defined policies.

Trusted Virtual Platforms are the core enabling component of our research around converged devices – client platforms such as notebooks or desktop PCs that can safely host multiple virtualized operating systems and virtual appliances concurrently and report accurately on the trustworthiness of the individually executing entities (Figure 1). We note that our work is largely applicable to the server side too but we have concentrated on the client side as this creates interesting user experience and performance

[3] TPMs ship on a large volume of motherboards from the leading PC manufacturers.

[4] This is known as *secure boot*. An alternative where measurements of the boot components are taken but the machine is still allowed to boot regardless of those measurements is more common and known as *authenticated boot*.

[5] Such as the BIOS, the boot loader and the operating system sitting under the application.

[6] For example, that the integrity of the client platform's boot loader hadn't been compromised by a virus.

[7] Such as network or video card emulation code.

challenges[8] that so far have not been adequately addressed by the research community.

The rest of the paper proceeds as follows: We start by giving some brief motivating examples for our pursuit of the Trusted Virtual Platform concept and highlight the key requirements that we are attempting to satisfy with our work. Then in the first of the two main sections of the paper we describe our work around restructuring the Xen control plane based on our core management interface and service architecture. In the second we describe our Trusted Virtual Platform concept and our extended integrity measurement architecture. We finish with a survey of related work and some concluding remarks hinting at areas of future work.

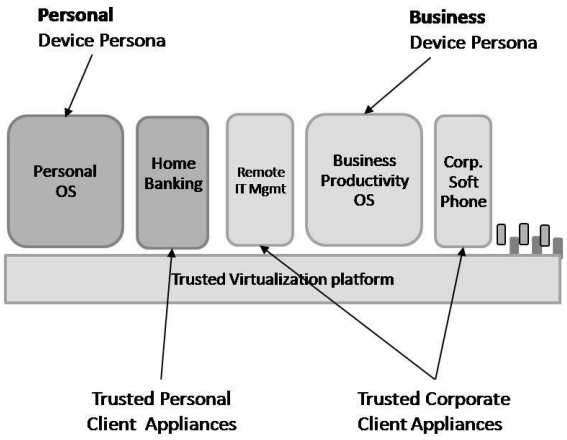

Figure 1: Trusted Converged Client

Much of the work described here has been done within the context of the European FP6 Open Trusted Computing (Open_TC) project and we thank our colleagues in that project for their collaboration. In particular we would like to acknowledge the work carried out by Carsten Weinhold whilst on placement in HP Labs from the Technical University of Dresden, Operating System Research Group.

2. USE SCENARIOS AND REQUIREMENTS

At a high level we can take two different views on an IT system to capture requirements and illustrate the motivations for our work in pursuing the Trusted Virtual Platform concept. Starting with an end-user view of an IT system, typically users want to know that they are using the application they expect to be using and to be confident that that application is operating free from interference. They may also want to run applications of differing trust levels safely on a single platform; for example, more and more of our daily lives rely on access to online services such as those provided by financial institutions. This makes it increasingly likely that a successful attack via untrustworthy software on a user's platform will result in access to sensitive or valuable information or assets that a user would rather have protected. The need for convenient containment environments for a user's applications is therefore also increasing.

IT operations, on the other hand, are looking for robust and effective means of easing the burden and difficulty of enforcing organizational security policy. In the cases where the boundary of an organization begins to stretch, such as where an organization makes provision for mobile and home working, organizations have a need to be able to determine the trust and security properties of an end-point device remotely before allowing it access to corporate resources. At a top level, IT operations are looking for strong assurance that their business IT systems are functioning when and as required.

The Trusted Virtual Platform concept can help from both of these viewpoints. From a user's perspective, the Trusted Virtual Platform concept gives a user the capability to run multiple applications of differing trust levels, say banking and gaming, on the same machine. It provides strongly isolated execution environments and the integrity and security of a properties of particular contained execution environment can be demonstrated to, say, an online bank where required. This brings benefits from an IT operations view too. We are increasingly seeing a blurring of the line between work and social lives with an associated increase in the shared use of IT resources for both business and personal activities. Just as the user can be assured that the business applications on their system cannot interfere with their personal applications and data, IT operations can be assured that their business applications and data can be hosted on an end user's platform safely isolated from anything else the end user may have (perhaps inadvertently) introduced onto their platform.

Even without the explicit need to support shared use of a platform for business and personal use, the Trusted Virtual Platform has value for an IT organization. As a simple example, end point computing platforms in use within an organization are commonly required to run some sort of personal firewall in order for the overall organization security policy to be upheld and enforced. In that model, the IT operations are quite sensitive to what else is running on an endpoint platform and are dependent upon the endpoint platform OS working as expected and not being under the control of an attacker, for example. This is hard to guarantee for a general-purpose endpoint platform OS such as Microsoft Vista or Linux. Using the Trusted Virtual Platform concept, the main endpoint platform OS can be run on one Trusted Virtual Platform, while security functionality such as a personal firewall can be run isolated away from the main OS on another Trusted Virtual Platform on the same physical machine. Underlying virtualization controls can be used to ensure that all network traffic from the main OS is routed through that firewall before it is allowed out onto the physical network. With this model, IT operations need only to be assured about the firewall Trusted Virtual Platform and underlying virtualization components; the network firewall controls will be applied no matter what state the main user OS is in and whether it is under the control of an attacker or not. The job of providing ongoing assurances about the overall health and well-being of the business IT systems can be eased greatly if certain assumptions like these about the end point platforms can be made and verified in a remote fashion.

From the scenarios painted above we can begin to extract some general requirements needed to support our Trusted Virtual Platform concept. Clearly, robust isolation of concurrent execution environments at the whole system level is required. The ability to apply mandatory policy controls over the contained

[8] For example, how to allow access to modern machine capability such as hardware accelerated graphics in a safe and trustworthy fashion.

execution environments is also required, such as the mentioned ability to dictate that network traffic from one virtualized operating system should be routed via another one before the traffic is allowed out onto the physical network. Finally, it needs to be possible for a remote party to establish the trust and security properties of a particular endpoint device in a meaningful fashion before granting access to sensitive or valuable resources.

3. TRUSTED VIRTUALIZATION LAYER

In this section we describe our work towards improving the trustworthiness of the Xen Hypervisor. We first describe our control plane disaggregation architecture. The aim of this is to partition and reduce the amount of privileged code forming Xen. This is based on splitting up the control plane normally provided by the Domain 0 operating system of Xen into a number of partitioned services. We then describe our library OS work which is a useful toolkit for hosting partitioned control plane services. Finally, we show how we can provide a high-performance disaggregated display virtualization service for our guest VMs with a secure GUI for managing trustworthy interactions with those VMs.

3.1 General Approach

Our approach is pretty simple in theory – namely, reduce the size and complexity of the code you have to trust in order for the isolation guaranties to be upheld. In practice things are a little more difficult. The first step in reducing the amount of privileged or trusted code on a system is establishing exactly where that code resides.

On Xen, virtual machines are known as *domains*. Guest operating systems run within a domain. Domain 0 is a privileged domain and is used to manage IO and the other domains on the system. The operating system running within domain 0 is known as the control-plane OS. A simplified representation of the Xen Hypervisor is shown in Figure 2. Here we can see four virtual machines running: Domain 0 which is hosting the control plane OS (typically a Linux distribution of some form), two normal guest OSes and another VM hosting a virtual appliance guest OS. A virtual appliance is simply a self-contained OS and application bundle.

The circled boxes in Figure 2 attempt to show that in order for isolation guarantees for guests to be upheld, we are trusting not only the core hypervisor layer (bottom circled box) but also the management and control plane OS running in Domain 0 (top left circle). For the purposes of this report we concentrate on the top left circle of the figure[9].

3.2 Control Plane Restructuring

Creating a trusted virtualization layer is a challenge because typically VMM layers require their control plane functionality to be hosted by a privileged VM running a general purpose OS such as Linux. This is Domain 0 under Xen. The control plane is normally structured in this fashion so as to enable the use of existing tools and applications for installing, configuring and managing the system. However, general-purpose operating systems have a large code base and potential attack surface. The challenge here is maintaining a balance between management convenience, management application support, performance and a minimal set of privileged code on the system.

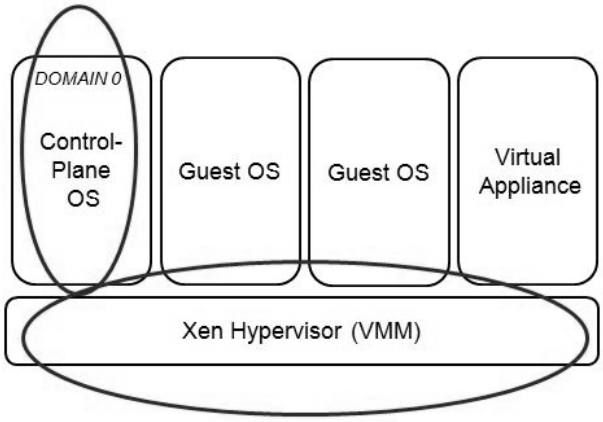

Figure 2: The Xen Hypervisor

The restructuring of the privileged Xen management and control plane we call *disaggregation* where essentially we identify the security critical features of the platform and move them into separate and smaller restricted privilege domains. Two good examples of such a restructuring include the development of a privileged Domain Builder VM (Domain B) [5] and a graphics domain (see section 2.3). These are two critical services provided by the platform and form part of the overall Trusted Computing Base (TCB). The TCB is what you rely on to uphold your security guarantees or properties. Our key contribution here is the architecture and development of a core privileged basic management engine that provides an appropriate interface (BMSI or Basic Management and Security Interface) for running less-privileged disaggregated management / control components on top of it (Figure 3).

Example functionality provided by the BMSI interface includes the ability to create a new VM and manage its lifecycle. The interface also allows access to basic physical TPM functionality, such as the protective store features of the TPM so that sensitive data and keys used by a service can be protected by the hardware TPM.

3.3 Library OS

To support the restructuring of control plane functionality into separately partitioned services we have developed what we call a *library OS*. The library OS gives us the ability to run an application program directly on top of Xen without requiring a full-blown OS underneath the application. Using the library OS instead of say a Linux-based OS to host an application reduces the amount of code being trusted by a factor of 10–100 (from 1–10M lines of code to 100K lines of code).

The library OS is made up of a cross-development environment and a set of libraries. The library OS allows C language application programs to be developed that will run directly on Xen. Figure 4 shows the application interfaces. The support for applications under our library OS is limited to the functions

[9] We have also carried out work in reducing the amount of code at the based VMM level by prototyping a modular VMM where only the actual functionality required is loaded.

available in the LibC and LibGloss libraries, although we have recently added C++ library support as well as enhancing the MiniOS core so that it can better support the hosting of device drivers. As an example we have prototyped a MiniOS based Ethernet driver. A port of existing C or C++ code to the library OS environment would require adding support for any libraries used by that application.

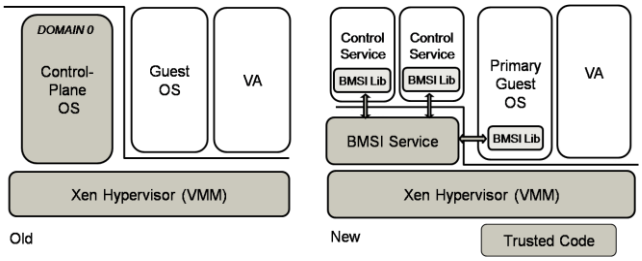

Figure 3: Restructured Xen Control Plane

Underneath these libraries sits a minimized kernel based on the Xen MiniOS example that forms part of the Xen distribution. We have also added an inter-domain communication mechanism for communication between library OS-hosted apps and apps in other guest operating systems. The application, library and kernel all run in a single address space within a single protection ring (ring 1 in the x86 version of Xen) so there is no distinction between kernel and library code. This is an appropriate choice for running small trusted services, but not general user code.

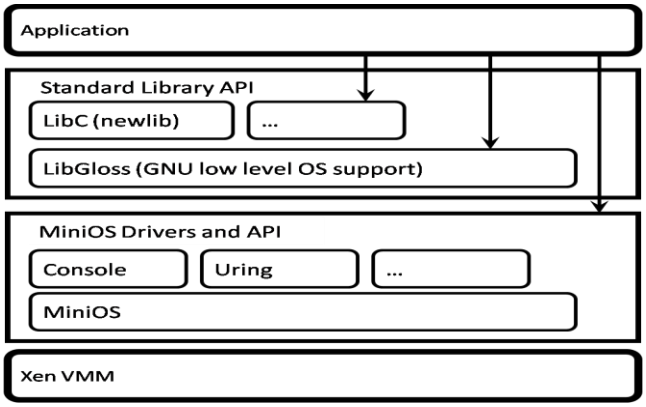

Figure 4: Xen Library OS

As well as C library support we also provide a development tool chain for actually building applications. The development tool chain is GNU-based allowing applications to be compiled for either Linux or library OS target environments from the same source code (Figure 5).

Servers built using the library OS need to be able to communicate with other domains. Conventional operating systems run as guests in Xen domains already have well-developed networking support, so it is possible to use protocols such as TCP/IP over virtual network interfaces for inter-domain communications.

For the small servers we expect to run over the library OS a full TCP/IP protocol stack adds unnecessary code size and complexity. Instead, we use the existing interface provided by Xen to implement inter-domain communications as a library component layered over the Xen shared memory mechanisms (Uring in Figure 4).

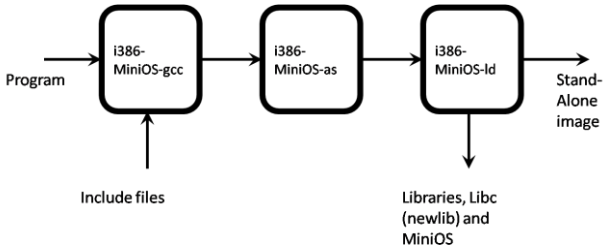

Figure 5: Library OS tool chain

3.4 Display Virtualization Service

The standard model for providing guest operating systems with graphical services under Xen is based on sharing the real physical machine video card via an X server running within Domain 0. Domain 0 accesses the real video card hardware directly; guest desktops are accessed from Domain 0 using either a VNC or SDL window hosted within the Domain 0 X server. Whilst this is a convenient method of accessing the guest desktops it has two major drawbacks.

First, the graphics performance and capability of desktops running over X server connections, whilst adequate for some server applications, is generally inadequate for client-side devices where high-performance and 3D graphical output is increasingly expected by end users. Research prototypes have started to appear that offer 3D capability for Xen guests[3] but support is limited to OpenGL primitives. The Aero graphical interface of Microsoft Vista is based on Direct X. Direct X cannot be directly mapped to OpenGL primitives so another approach is needed in order to support Vista.

Second, an X server is typically a large body of complex code. It is not ideal for this code to be hosted within the Domain 0 privileged VM. A compromise would give an attacker a pathway into accessing the privileged interface of Domain 0.

This next section looks at the work we have done in three areas: First, a more secure method of providing shared graphical services for guests (outside of Domain 0). Second, how to support the hardware acceleration and 3D interfaces required of the latest generation Microsoft operating systems. Third, we describe aspects of a GUI for securely managing user interactions with guest operating systems.

3.4.1 Graphics Disaggregation

Newer Intel and AMD hardware platforms[10] offer an IOMMU capability alongside the well-established CPU MMU capability.

With an IOMMU, a PCI card such as a graphics card can be securely directly assigned to a guest virtual machine[11]. The guest

[10] Intel Montevino/Cantiga for example.

[11] There are technical challenges in particular with pass-through graphics cards largely due to the need to support VBIOS operation. We have satisfactorily addressed those challenges on the HP 6930p notebook and DC7800 desktop platforms.

runs the real device drivers for the pass-through card and all the card functionality such as GPU hardware acceleration and 3D effects are available to it. The IOMMU is given a set of page tables that map the device's view of memory pages (guest pages) to actual physical pages (machine pages). In this way it is possible to limit the ability of a rogue guest OS being able to use the DMA capabilities of a pass-through device to compromise the system as a whole (such as overwriting the Xen memory area or reading another guest's memory area). In itself, an IOMMU doesn't allow a single hardware device to be shared[12] but does give us an important foundation on which to build.

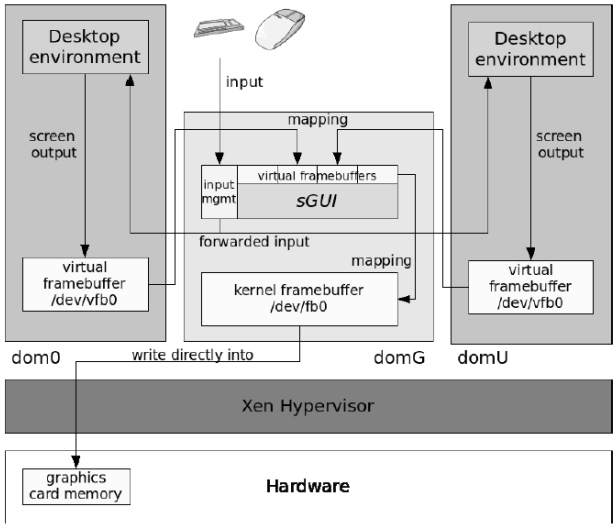

Figure 6: Simple Graphics Disaggregation

Figure 6 shows an initial first-step towards graphics display disaggregation. Here a separate domain (Domain G) hosts a virtual frame buffer mapping component. The physical machine video card is directly mapped to Domain G using an IOMMU. Other guests are provided with a virtual frame buffer device which simply forwards all the frame buffer reads and writes to the mapping service within Domain G. Domain G multiplexes the virtual guest frame buffers onto the real graphics device. This provides a simple graphics service for guests that doesn't rely on guests accessing the real graphics hardware through an X server in Domain 0.

3.4.2 GPU Virtualization

To support more than a simple virtualized frame buffer interface and offer guests access to hardware acceleration and 3D GPU features we have virtualized the Gallium[13] driver framework [8]. Gallium is a new architecture for providing 3D graphics drivers which is designed to be OS- and graphics hardware-independent.

[12] PCI-IOV [9] devices do help with this but IOV graphic cards are still somewhere off.

[13] Gallium is the driver architecture adopted by Intel for their Open Source graphics drivers.

Figure 7 shows the non-virtualized Gallium architecture. Gallium achieves portability across hardware GPUs and different OS interfaces for graphics (DirectX and OpenGL) by providing at a top level an abstract GPU instruction set that gallium components lower down in the stack map onto the real GPU instruction set. We add virtualization to the Gallium stack above its abstract GPU instruction set interface.

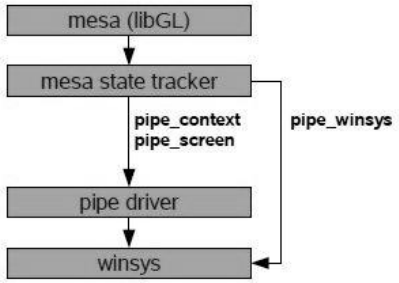

Figure 7: Gallium 3D Driver Architecture

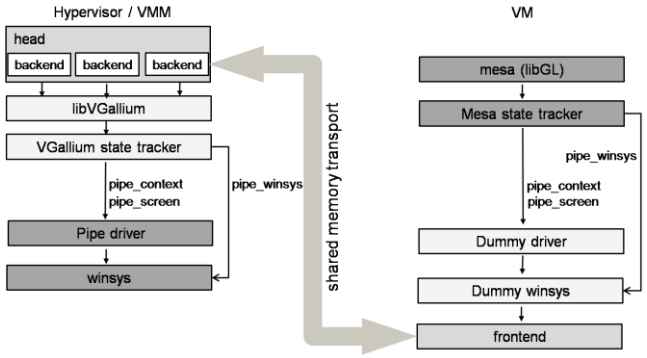

Figure 8: Virtualized Gallium 3D Architecture

The key part of our virtualized Gallium stack is a virtualized Gallium driver *head* – the component shown in Figure 8 as hosting multiple "backends." This is responsible for initializing the real Gallium driver, assigning defined screen regions and buffers to guests, and responding to init requests from guests (screen size or graphics mode for example). It also acts as a dispatcher that connects and disconnects specific guest command streams to and from the real driver. By plugging the virtualized gallium stack into our disaggregated graphics domain we can safely provide guests with access to the hardware acceleration and 3D GPU features of the physical graphics card.

3.4.3 Secure GUI service

The virtualized Gallium driver architecture gives us additional functionality such as the ability to do command stream filtering and translation. It also allows compositing such as the transparent layering of guest windows (alpha blending). With this support we can provide a secure GUI for both allowing guest windows to safely share a single screen area and to provide Guest VM graphical management functionality such as the ability to query VM states, start and stop VMs, or grab / filter input events, for

example. Other features include the potential for guest window labeling (e.g. to indicate security states).

4. TRUSTED VIRTUAL PLATFORM

Under a VMM or hypervisor-based system, a guest OS runs on top of what is traditionally referred to as a "virtual platform" (Figure 9). A virtual platform is composed of a set of virtual devices (such as virtual CPU, memory, disks, network, video, etc.) interconnected according to a specific layout and accessed by a guest OS running on top of a Virtual Machine (VM) interface. The layout and the security configuration of each virtual device are what define the identity and integrity of a virtual platform.

To ease the burden associated with managing the security lifecycle of a guest OS running on a virtualized system, it is important that the integrity of its virtual platform is maintained and the desired security properties enforced, and this is where we introduce the notion of a "trusted" virtual platform (or TVP).

A TVP is a virtual platform where we can guarantee and report on the integrity and security properties of that virtual platform. One aim of our trusted virtual platform concept is to achieve the same level of binding between virtual platform components that there is on a physical platform.

To support the notion of a TVP we provide a Virtual Platform Enforcement Service (VPES) to interpose on the normal VM lifecycle management actions. When a request for a new VM is initiated by the management system, the VPES creates two additional VMs that are attached to the original VM (Figure 10). One VM is the Virtual Device Protection Service which is responsible for enforcing information flow policies for communications to and from the Virtual Platform. The second one is a Virtual TPM which is used to identify the Virtual Platform. It can also be used by the VM itself to enforce some intra-OS security policies, just as a conventional physical TPM could be used by a non-virtualized operating system.

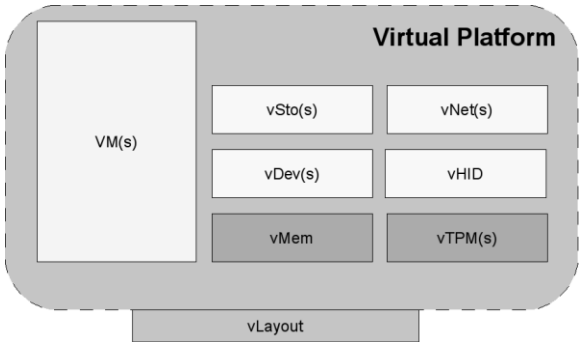

Figure 9: Virtual Platform Architecture

The VPES, VDPS and VTPMs interact together through the Basic Management and Security Interface (BMSI) as introduced in 3.2. This interface provides a minimal set of security and integrity primitives required to bootstrap each security service and implement the runtime integrity model of the Virtual Platforms.

For Xen fully-virtualized (HVM) guest operating systems the majority of the hardware devices seen by that guest operating system are provided by emulated device models of real hardware devices. Under the latest versions of Xen, all device model code for a particular HVM domain is hosted within a stub-domain[14]. This is the DM component in Figure 10. In our TVP model, the VDPS service verifies the integrity of that DM domain before allowing the guest operating system to begin execution.

5. INTEGRITY MEASUREMENT AND REPORTING ARCHITECTURE

The virtualization-specific integrity measurement and reporting architecture described in this section makes it possible for the trust and security properties that we achieve at the trusted virtualization layer and the trusted virtual platform level to be attested[15] to and remotely verified. It is an important requirement for us that the integrity and reporting architecture manages to capture all the dependencies of a guest operating system.

The core of this integrity and reporting framework is the hardware TPM. The TPM allows us to root the trust and security properties of our trusted virtualization layer in hardware. Hardware-based trust and security mechanisms offer immunity from the wide class of software-based attacks. The TPM thus provides a good foundation for the overall system trust and security properties.

The disaggregation of the VMM control plane described in Section 3 reduces the amount of trusted code on the system, giving a reduced set of components we need to measure and report on, allowing for more meaningful verification. Generally, the smaller the component you can measure the better. This is based on the assumption that if you are measuring a sufficiently small component then you can have more confidence in its predicted behavior compared to a larger component.

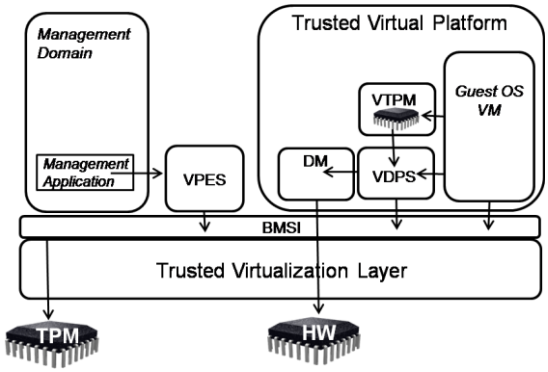

Figure 10: Trusted Virtual Platform Components

Using TCG terminology [10], we build a chain of trust from the hardware TPM to our Trusted Virtualization Layer[16]. The basic management and security interface (BMSI) of Section 3 exposes primitive TPM functionality for use by other platform services.

[14]Stub-domains are almost identical to our Library OS.

[15]This is the TCG term for the process of vouching for the accuracy of information. A TPM *attests* to the contents of its PCR registers.

[16]On our latest platforms we use Intel TXT and AMD SVM dynamic root of trust mechanisms to get our virtualization layer up and running in a measured state.

The component implementing the BMSI is included as part of the trusted virtualization layer measurements, as this component is designed to encapsulate most of the core privileged code on the system above the actual VMM layer.

We augment the trusted virtualization layer measurements with measurement of the Virtual Platform Enforcement Service (VPES), the component described in Section 4 that we use to interpose on all the VM lifecycle management operations as part of the Trusted Virtual Platform concept. This forms the basis of our static chain of trust, which is used to report on the integrity of the dynamic components and configuration that make up the virtual platforms.

Reporting of integrity measurements to a remote party during its interaction with an application in a guest OS happens in much the same way as on a non-virtualized platform: a *chain of trust* in the (virtual) platform is built up so that when interacting with a particular application on a (virtual) platform, a report can be obtained about the components on the (virtual) platform. The remote party, for example an online banking service, can use this report to establish the trustworthiness of a particular user's client (virtual) platform before allowing it access to a service. However, in the case of a virtual platform, the integrity report needs to capture all of the dependencies of the virtual platform, i.e., its underlying components such as the VMM (Trusted Virtualization Layer).

The model we present allows this report to be conveyed to a third party in two ways. With one method, the VM can make use of its Virtual TPM (in the same standard way it would use a hardware TPM) to obtain a digital signature of its current integrity. Because of the way the VTPM is implemented (using the Virtual Device Protection Service), such a signature will contain an implicit attestation of the current integrity of the TCB as well as the integrity of the Virtual Platform (its device configuration).

An alternative method requires a special type of device (also implemented by using the VDPS) which responds to a remote party as part of a specific protocol before data is allowed to pass in and out of a virtual platform. This could be implemented as part of the Virtual Platform network device. The enhanced network device would implement an attestation response and establish an encrypted channel with the remote party before the guest OS is granted network access. With such a solution the guest OS doesn't need to be aware of or involved in the protocol for the attestation, which is useful in supporting legacy applications.

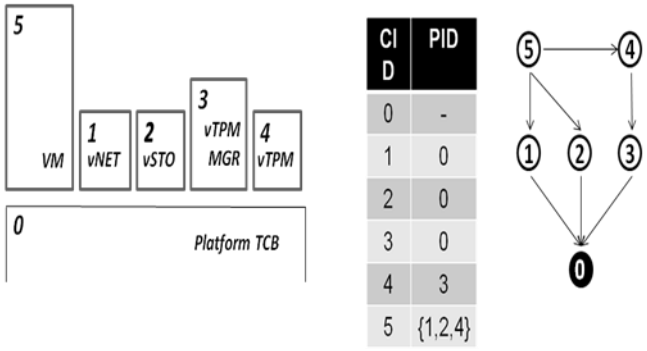

Figure 11: Measurement and Reporting Framework

An important aspect of the integrity and measurement framework is that it allows peer as well as strict hierarchical relationships between components to be captured via integrity measurements. For example, in Figure 11 the main VM of a virtual platform (component 5) depends on components 1, 2 and 4 even though from a virtualization layer perspective they are all peers. Following the traditional TCG measurement framework the relationship between component 5 and the others could not be captured.

6. RELATED WORK

Berger et al [2] present the concept of a Virtual TPM. We believe their current implementation doesn't sufficiently capture the state of all the dependencies of a guest OS. Disaggregation of the Xen domain builder is discussed in [5]. Our BMSI architecture presents a more abstracted approach to disaggregation.

7. SUMMARY

This paper has described our architecture for enforcing integrity and security policy controls over a guest operating system or virtual appliance running on top of a virtual platform using our Trusted Virtualization Layer and Trusted Virtual Platform concepts. The architecture also allows for attestation and remote verification of the virtualized platform trust and security properties via our virtualization-specific integrity measurement and reporting architecture. The measurement architecture is designed to reflect all the dependencies of the virtual environment of a guest operating system. This allows remote parties to accurately gain confidence in the (virtual) clients with which they are interacting.

8. REFERENCES

[1] Barham, P., et al. 2003. Xen and the Art of Virtualization. Proc. 19th ACM Symp. operating systems Principles (SOSP 03), pp. 164-177.

[2] Berger, S., et al. 2006. vTPM: virtualizing the trusted platform module. USENIX Security Symposium. pp. 21-21.

[3] Lagar-Cavilla, H.A., 2007. VMM-independent graphics acceleration. ACM/Usenix International Conference On Virtual Execution Environments. pp. 33-43.

[4] McConnell, Steve. 1993. Code Complete. Microsoft Press.

[5] Murray, D. 2008. Improving Xen security through disaggregation. Proceedings of the Fourth ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments.

[6] Microsoft. Microsoft Bitlocker Drive Encryption. http://technet.microsoft.com/en-us/windows/aa905065.aspx.

[7] Trusted Computing Group. http://www.trustedcomputinggroup.org.

[8] Tungsten Graphics. Gallium 3D. 2008. http://www.tungstengraphics.com/wiki/index.php/Gallium3D.

[9] PCI-SIG Specifications. PCI-SIG IO Virtualization. http://www.pcisig.com/specifications/iov/

[10] Balacheff, B., et al. 2002. Trusted Computing Platforms. Prentice Hall, 2002.