Terra: A Virtual Machine-Based Platform for Trusted Computing

Tal Garfinkel Ben Pfaff Jim Chow Mendel Rosenblum Dan Boneh
{talg,blp,jchow,mendel,dabo}@cs.stanford.edu
Computer Science Department, Stanford University

ABSTRACT

We present a flexible architecture for trusted computing, called Terra, that allows applications with a wide range of security requirements to run simultaneously on commodity hardware. Applications on Terra enjoy the semantics of running on a separate, dedicated, tamper-resistant hardware platform, while retaining the ability to run side-by-side with normal applications on a general-purpose computing platform. Terra achieves this synthesis by use of a *trusted virtual machine monitor* (TVMM) that partitions a tamper-resistant hardware platform into multiple, isolated virtual machines (VM), providing the appearance of multiple boxes on a single, general-purpose platform. To each VM, the TVMM provides the semantics of either an "open box," i.e. a general-purpose hardware platform like today's PCs and workstations, or a "closed box," an opaque special-purpose platform that protects the privacy and integrity of its contents like today's game consoles and cellular phones. The software stack in each VM can be tailored from the hardware interface up to meet the security requirements of its application(s). The hardware and TVMM can act as a trusted party to allow closed-box VMs to cryptographically identify the software they run, i.e. what is in the box, to remote parties. We explore the strengths and limitations of this architecture by describing our prototype implementation and several applications that we developed for it.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

design, security

Keywords

virtual machine, virtual machine monitor, VMM, trusted computing, attestation, authentication

1. INTRODUCTION

Commodity computing systems have reached an impasse. There is an increasing need to deploy systems with diverse security requirements in enterprise, government, and consumer applications. However, current hardware and operating systems impose fundamental limitations on the security these platforms can provide.

First, commodity operating systems are complex programs that often contain millions of lines of code, thus they inherently offer low assurance. Building simple, high-assurance applications on top of these operating systems is impossible because applications ultimately depend on the operating system as part of their trusted computing base.

Next, commodity operating systems poorly isolate applications from one another. As a result, the compromise of almost any application on a platform often compromises the entire platform. Thus, applications with diverse security requirements cannot be run concurrently, because the platform's security level is reduced to that of its most vulnerable application.

Further, current platforms provide only weak mechanisms for applications to authenticate themselves to their peers. There is no complete and ubiquitous mechanism for distributed applications to verify the identities of programs they interact with. This makes building robust and secure distributed applications extremely difficult, as remote peers must be assumed to be malicious. It also significantly limits the threat models that can be addressed. For example, an online game server cannot tell whether it is interacting with a game client that will play fairly or one which has been subjected to tampering that will allow users to cheat.

Finally, current platforms provide no way to establish a trusted path between users and applications. For example, an application for trading on financial markets has no way of establishing if its inputs are coming from a human user or a malicious program. Conversely, human users have no way of establishing whether they are interacting with a trusted financial application or with a malicious program impersonating that application.

To address these problems, some systems resort to specialized closed platforms, e.g. cellular phones, game consoles, and ATMs. Closed platforms give developers complete control over the structure and complexity of the software stack, thus they can tailor it to their security requirements. These platforms can provide hardware tamper resistance to ensure that the platform's software stack is not easily modified to make it misbehave. Embedded cryptographic keys permit these systems to identify their own software to remote systems, allowing them to make assumptions about the software's behavior. These capabilities allow closed platforms to offer higher assurance and address a wider range of threat models than current general-purpose platforms.

The security benefits of starting from scratch on a "closed box" special-purpose platform can be significant. However, for most applications these benefits do not outweigh the advantages of general-purpose open platforms that run many applications including a huge body of existing code and that take advantage of commodity hardware (CPU, storage, peripherals, etc.) that offers rich functionality and significant economies of scale. In this work, we describe a software architecture that attempts to resolve the conflict between these two approaches by supporting the capabilities of closed platforms on general-purpose computing hardware through a combination of hardware and operating system mechanisms.

Our architecture, called Terra, provides a simple and flexible programming model that allows application designers to build secure applications in the same way they would on a dedicated closed platform. At the same time, Terra supports today's operating systems and applications. Terra realizes this union with a *trusted virtual machine monitor* (TVMM), that is, a high-assurance virtual machine monitor that partitions a single tamper-resistant, general-purpose platform into multiple isolated virtual machines. Using a TVMM, existing applications and operating systems can each run in a standard virtual machine ("open-box VM") that provides the semantics of today's open platforms. Applications can also run in their own closed-box virtual machines ("closed-box VMs") that provide the functionality of running on a dedicated closed platform. The TVMM protects the privacy and integrity of a closed-box VM's contents. Applications running inside a closed-box VM can tailor their software stacks to their security requirements. Finally, the TVMM allows applications to cryptographically authenticate the running software stack to remote parties in a process called *attestation*.

Both open- and closed-box VMs provide a raw hardware interface that is practically identical to the underlying physical machine. Thus, VMs can run all existing commodity software that would normally run on the hardware. Because a hardware-level interface is provided, application designers can completely specify what software runs inside a VM, allowing them to tailor an application's software stack to its security, compatibility, and performance needs. Closed-box VMs are isolated from the rest of the platform. Through hardware memory protection and cryptographic protection of storage, their contents are protected from observation and tampering by the platform owner and malicious parties.

The next section presents the Terra architecture and describes the basic properties of trusted virtual machine monitors, the mechanism that allows applications with open-box and closed-box semantics to run side-by-side. It describes the process of attestation that Terra uses to identify the contents of VMs to remote parties and presents several models for user interaction with the TVMM. Section 3 describes Terra's local security model, Terra's impact on application assurance, and how remote parties can take advantage of Terra's security model. In Section 4 we describe the design of the Terra TVMM. Section 5 describes a prototype implementation of this design and the implementation of closed-box applications that utilize our prototype. These applications include a "cheat-resistant" closed-box version of the popular multi-player game Quake and trusted access points (TAPs), a system of closed-box VMs that can be used to regulate access to a private network at its endpoints. We discuss related work in section 6 and conclude in section 7.

2. TERRA ARCHITECTURE

At the heart of Terra is a virtual machine monitor (VMM). Like any VMM, Terra virtualizes machine resources to allow many virtual machines (VMs) to run independently and concurrently. Terra also provides additional security capabilities including acting as a trusted party to authenticate the software running in a VM to remote parties. Because of this property we refer to it as a "trusted VMM" (TVMM).

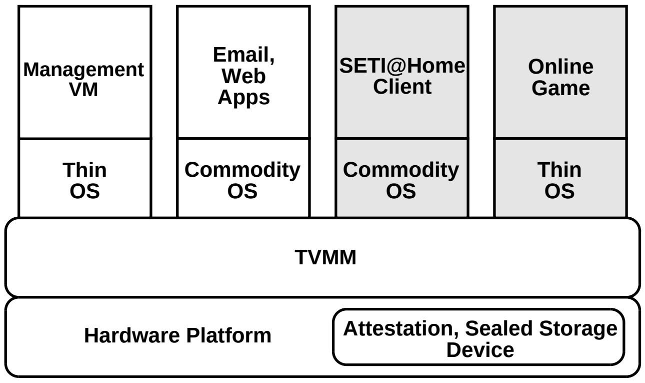

Figure 1: Terra Architecture. A trusted virtual machine monitor (TVMM) isolates and protects independent virtual machines (VMs). Closed box VMs, shown in gray, are protected from eavesdropping or modification by anyone but the remote party who has supplied the box. Here, the SETI@Home client is in a closed box so that its server can verify that it has not been modified to claim it has run checks that it actually has not, and an online game is in another to deter cheating (see section 5.2 for more information). The TVMM can identify the contents of the closed box to remote parties, allowing them to trust it. Also shown here are the management VM and an open-box VM running a commodity operating system.

At a high level, the TVMM exports two VM abstractions. *Open-box VMs* provide the semantics of today's open platforms. These can run commodity operating systems and provide the appearance of today's general-purpose platforms. *Closed-box VMs* implement the semantics of a closed-box platform. Their content cannot be inspected or manipulated by the platform owner. Thus, their content is secure, neither inspectable nor modifiable by any but those who constructed it, who can explicitly provide themselves access. Figure 1 depicts an instance of the Terra architecture with an open-box VM, two closed-box VMs, and the management VM (to be described later).

Terra provides a raw virtual machine as the development target for applications, lending great flexibility to application designers. Applications can be designed from the (virtual) hardware up, using the operating systems that best suit their security, portability, and efficiency needs. Operating systems that run in VMs may be as simple as a bootstrap loader plus application code or as complex as a commodity operating system that runs only one application. Applications can completely tailor the OS to their security needs. Instead of running single closed-box applications, a closed-box VM might run a special trusted OS with a selection of applications designed specifically for it, thus providing something similar to the NGSCB [13] model.

VMs on a single physical machine communicate with one another over virtualized standard I/O interfaces such as NICs, serial ports, etc. The VMM can also multiplex the display and input devices. Thus, from the user's perspective, a closed-box VM may take on the appearance of a normal application, a virtual network appliance, or a virtual device (e.g. a USB device).

The responsibility for configuring how these VMs are granted storage and memory, connected, started, stopped, etc. is delegated to a special *management VM*. The TVMM offers the management VM a basic interface to carry out these tasks. Where the TVMM provides resource management mechanisms, the management VM decides policy, providing a higher-level interface to users and other VMs.

2.1 The Trusted Virtual Machine Monitor

Terra's architecture is based on a virtual machine monitor [33], a thin software layer that allows multiple virtual machines to be multiplexed on a single physical machine. The virtual machine abstraction that the VMM presents is similar enough to the underlying architecture that programs and operating systems written for the physical hardware can run unmodified on the virtual hardware. Terra takes advantage of the following properties of traditional VMMs:

Isolation A VMM allows multiple applications to run in different virtual machines. Each virtual machine runs in its own hardware protection domain, providing strong isolation between virtual machines. Secure isolation is essential for providing the confidentiality and integrity required by closed-box VMs. Also, the abstraction of separate physical machines provides an intuitive model for understanding the isolation properties of the platform.

Extensibility Any "one size fits all" approach to providing an operating system for a trusted platform greatly limits a platform's flexibility because it ties all applications to one interface. If this interface is too complex, it compromises the simplicity of the system, forcing many applications to deal with an unacceptably low level of assurance. Conversely, if it is too simple, it compromises the performance and functionality of the system, severely limiting the variety of applications that can usefully take advantage of it.

Terra addresses this conflict by allowing application implementers to view a VM as a dedicated hardware platform, allowing an application's software stack to be built from the (virtual) hardware up. This allows application designers to select the OS that best addresses their requirements for security, compatibility, and performance.

For example, simple applications that require very high assurance (e.g. electronic voting) can use a very minimal OS layer that consists of little more than bootstrapping code. Other applications (e.g. the trusted access points covered in section 5.3) may require high assurance and a rich set of OS primitives for access control such as the NSA's SELinux [43] or EROS [52]. A third class of applications may need only a modest level of assurance, but require a relatively feature-rich OS offering high performance and compatibility, such as a stripped-down version of Windows or Linux. Many online games likely fall in this category.

Beyond choosing an operating system to meet their needs, designers can tailor the OS to include only the components required for their applications. Modular OSes such as QNX and Windows CE, commonly used in embedded systems, illustrate how an OS can facilitate this type of application-specific customization.

Efficiency Experience with virtual machine monitors over the past 30 years has shown the overhead of virtualization on virtualizable hardware platforms can be made essentially negligible. Even without virtualizable hardware, the overheads can be made very small [34]. Thus, a VMM can provide essentially the same properties as separate devices with more modest resources whose total resources sum to those of the physical machine. An application running under Terra can potentially be more efficient than its standard OS counterpart because it can tailor the OS abstractions it uses to its needs as in exokernels [24]. This is essential to providing a platform flexible enough to run a wide range of applications with differing performance demands.

Compatibility VMMs can run today's operating systems, such as Linux and Windows, and applications without modifications, unlike alternative approaches to secure isolation, such as microkernels [42] and isolation kernels [61]. This allows existing systems to run under Terra, and means that specialized standalone applications targeted to Terra can run side-by-side with legacy applications. The greater isolation of a VM, compared to a process in an ordinary OS, can improve assurance on its own; untrusted applications can be transformed into low-assurance trusted applications in closed boxes with minimal changes (see section 5.2 for an example). It also allows new stand-alone applications to leverage existing toolchains, operating systems, etc. for their construction.

Security A VMM is a relatively simple program (Disco has only 13,000 lines of code [12]), with a narrow, stable, well-defined interface to the software running above it. Unlike traditional operating systems, that must support filesystems, network stacks, etc., a VMM only needs to present relatively simple abstractions, such as a virtual CPU and memory. As a result of these properties, VMMs have been heavily studied as an architecture for building secure operating systems [38, 30]. VMMs have long been a mainstay of mainframe computing [20], where their security has been leveraged for implementing systems for banking and finance, health care, telecommunication [1], defense [47], etc. The isolation properties of real-world VMMs such as that of the IBM zSeries have received intense scrutiny and been certified as conforming to the highest standards for assurance according to Common Criteria requirements [3].

Terra's TVMM provides three additional capabilities not found in traditional VMMs. These capabilities are essential to providing a "closed box" abstraction:

Root Secure Even the platform administrator cannot break the basic privacy and isolation guarantees the TVMM provides to closed-box VMs.

Attestation This feature allows an application running in a closed box to cryptographically identify itself to a remote party, that is, to tell the remote party what is running inside the closed box. This allows that party to put trust in the application, i.e. to have faith that the application will behave as desired. The following section discusses the basics of attestation.

Trusted Path Providing a *trusted path* from the user to the application is essential for building secure applications [44]. In a TVMM, a trusted path allows a user to establish which VM they are interacting with as well as allowing a VM to ensure that it is communicating with a human user. It also ensures the privacy and integrity of communications between users and VMs, thereby preventing snooping or tampering by malicious programs.

2.2 Attestation and VM Identity

Attestation enables an application in a VM to authenticate itself to remote parties [39, 62]. Attestation authenticates who built the platform hardware and what software was started at each layer of the software stack, from the firmware up to the VM. Receiving an attestation tells the remote party what program was started on a platform, but it does not confirm that the program has not subsequently been compromised. The party receiving an attestation must judge for itself how strongly it believes in the correctness and security of each of the platform's layers.

Attestation requires building a certificate chain, from the tamper-resistant hardware all the way to an application VM, to identify each component of the software stack. This chain begins with the hardware, whose private key is permanently embedded in a tamper-resistant chip and signed by the vendor providing the machine. The tamper-resistant hardware certifies the system firmware (e.g. PC BIOS). The firmware certifies the system boot loader, which certifies the TVMM, which in turn certifies the VMs that it loads.

At a high level, each certificate in this certificate chain is generated as follows: A component on the software stack that wants to be certified first generates a public/private key pair. Next, the component makes an ENDORSE API call (see section 4.3) to the lower level component, passing its public key and possibly other application data it wants certified. The lower-level component then generates and signs a certificate containing (1) a SHA-1 hash of the attestable parts of the higher-level component, and (2) the higher-level component's public key and application data. This certificate binds the public key to a component whose hash is given in the certificate.

Certification of a VM being loaded by the TVMM involves the TVMM signing a hash of all persistent state that identifies the VM. This includes the BIOS, executable code, and constant data of the VM. This does not include temporary data on persistent storage or NVRAM contents that constantly change over time. The separation between data which does and does not need to be included in the attestation is application-specific, made by the VM's developer. Terra supports these two type of data by providing VMs with both "attested storage" that the TVMM incorporates in the VM's hash and "unattested storage" that it does not (see section 4.2).

Example attestation

As an example of how a VM can use an attestation certificate, consider a home banking application VM, such as Quicken, that is attesting its validity to a remote banking server. For simplicity, we assume that the VM and remote server are establishing an authenticated channel using the standard SSL session key exchange protocol. SSL is well suited for this purpose because it allows both parties the opportunity to present a certificate chain.

For the SSL handshake protocol the VM and remote party use their attestation certificate chains and private keys for authentication. At the end of the protocol both parties share a secret session key. During the handshake protocol, the remote server validates the VM's certificate chain as follows:

1. It verifies that the lowest certificate in the chain, certifying the hardware, is from a trusted certificate authority and that the certificate has not been revoked.

2. It verifies that all hashes in the certificate chain are on the remote server's list of authorized software. That is, the remote server trusts the BIOS, the bootloader, and the TVMM.

3. It verifies that the hash of the VM's attested storage, provided in the topmost certificate, is on a list of authorized applications (e.g. the VM is a valid version of Quicken).

If all these checks are satisfied, then the remote server knows that it is communicating with an authorized application VM. It then completes the session-key exchange protocol to establish an authenticated channel. Omitting the key exchange would open up the attestation process to a man-in-the-middle attack. For example, a malicious user could wait for attestation to complete, then reboot the machine into an untrusted state without the remote server's knowledge.

Establishing trust

Validation of the VM's attestation certificate chain at the remote server requires further explanation. In the discussion above we required the remote server to verify that the hash of the VM's attested storage is on the server's list of authorized applications. However, since there are many versions of a given application it is unreasonable to require the remote server (e.g. a bank) to keep track of hashes of all these versions. Instead, the remote server should require the application VM to also send a certificate from its software vendor (e.g. Intuit, in the case of Quicken) certifying that a given VM hash is indeed a valid version of the application. Thus, we see that the attestation certificate chain proves to the remote server the components that were loaded onto the local machine. The remaining certificates prove what these components are.

We can see from the above that two chains of trust are involved in attestation. Both of these start at a CA (either the same CA or different ones) and end at the application VM. The first chain certifies that a particular software binary image is running; the CA certifies the hardware manufacturer, which signs the tamper-resistant hardware, which signs the TVMM, which signs the application VM's hash. The second chain certifies that the binary image is in fact a version of some interesting program, e.g. version 4.3 of Quicken; the CA certifies the software manufacturer, which signs the VM's hash. Taken together, the two certificate chains show that a VM with a particular hash is running and that that hash represents a particular version of a particular software program. Additional chains, provided by the software vendors that shipped these components, can be used to certify the BIOS, boot loader, and TVMM. During attestation, all the certificates of interest are sent to the remote server, which uses them to decide whether it trusts the various software vendors and whether it trusts the applications that these software vendors are certifying.

Software upgrades and patches

The mechanism described above makes the software upgrade and patch process straightforward. Every upgrade to a VM simply includes a new certificate proving that the resulting VM hash is still a valid version of the application. Cumulative patches that supersede all previously released patches can work the same way.

The situation is a bit more complex for vendors that allow any subset of a collection of patches to be applied to a base VM. We speculate that in this case, the vendor could issue a certificate that states that a specified base VM, plus any or all of a list of specified patches, is a valid version of some software program. The TVMM would sign a certificate that identified the variant in use and include both certificates in attestations.

Revocation

A user who could extract the private key from the tamper-resistant hardware could completely undermine the attestation process. Such a user could convince a remote peer that the local machine is running well behaved software, when in fact it is running malicious code. Worse yet, by widely publishing the private key and certificate the user could enable anyone to undermine the attestation process.

This scenario shows the importance of revoking compromised hardware. Revocation information must be propagated to every host that might depend on revoked certificates for attestation using CRLs, OCSP, or CRTs (see survey of certificate revocation methods in [37]). It is much harder to recover from a compromise of a manufacturer's signing key (e.g. Dell's signing key) without re-certifying all deployed devices, so it is critical that manufacturers' private keys be protected as carefully as root CA private keys.

Privacy

The attestation process completely identifies the machine doing the attestation, which raises a privacy concern. Given the resistance met by Intel when it introduced processor serial numbers, this concern must be taken seriously.

One option for maintaining user privacy, proposed by the Trusted Computing Group, is to use a special CA, called a Privacy CA (PCA). Periodically, the user's machine sends an attested certificate request to a PCA. The PCA verifies the machine's hardware certificate and then issues a certificate containing a random pseudonym in place of the real identity. From then on, the machine uses this anonymized certificate for attestation. Although the mapping between real identities and pseudonyms is kept secret, the PCA does keep track of the mappings for revocation purposes. Note that the anonymized hardware certificate must be periodically renewed with a fresh pseudonym; otherwise the anonymized certificate functions as a unique processor ID.

Past experience shows that users are generally unwilling to pay for anonymity services such as PCAs. As a result, the PCA incurs significant liability with no income—not a good business model. Consequently, it is unlikely that this PCA mechanism will be used in practice.

Fortunately, practical cryptographic techniques enable private attestation without the need for a third party. The simplest mechanism, due to Chaum [14], is known as *group signatures*. A practical implementation is given in [9]. In our context, group signatures enable private attestation without any extra work from the user. When using group signatures, the hardware manufacturer embeds a different secret signing key in each machine. As in standard attestation, this key is used to sign the firmware (e.g. BIOS) at boot time. However, the signature does not reveal which machine did the signing. In other words, the attestation signature convinces the remote party that the hardware is certified, but does not reveal the hardware identity. Furthermore, in case a machine's private key is exposed, that machine's signing key can be revoked so that attestation messages from that machine will no longer be trusted.

Interoperability and Consumer Protection

Attestation is a valuable primitive for building secure distributed systems. It frequently simplifies system design and reduces protocol complexity [28]. However, attestation also has a variety of potentially ominous implications that bear careful consideration.

In today's open distributed systems, programs from any source can interoperate freely. This has led to a proliferation of clients and servers for a wide variety of protocols, including commercial, free, and open source variants. This has benefited consumers by fueling innovation, encouraging competition, and preventing product lock-in. Attestation would allow software vendors to create software that would only interoperate with other software they had provided. This creates the tremendous risk of stifling innovation and enabling monopoly control [5]. Given this risk it is critical that the deployment of attestation be given careful consideration, and that appropriate technical and legal protections are put in place to minimize abuse.

Another far-reaching implication of attestation is its ability to facilitate digital rights management (DRM). If trusted computing is deployed ubiquitously, media providers could decide to only release their content to platforms that would prevent copying, expire the media after a certain date or number of viewings, etc. A full discussion of technical, commercial, and legal implications are beyond the scope of this work.

2.3 Secure User Interface

A secure user interface provides a trusted path to applications. It prevents malicious applications from confusing the user about which VM is in use. This can be achieved by providing unforgeable and unobstructable visual cues that allow the user to identify the current VM. A wide range of options exist for addressing this from a UI design perspective.

One model presented by the NetTop architecture [47] is a virtual KVM (keyboard, video, mouse) switch model. In this model the user is presented with separate virtual consoles which the user can select using a virtual KVM switch. A small amount of space at the top of the screen displays which VM the physical console is currently showing. This space is reserved for exclusive use by the VMM.

Another option for accomplishing the same end has been used in compartmented mode workstation systems [18]. In these systems a secure window manager controls the entire desktop and applications (in our case, VMs) can only write to portions of the display to which they have been granted access. Tags on the frame of each window indicate which VM owns it, and a dedicated space is again reserved to inform the user which VM is in use.

We have not implemented a secure user interface in our Terra prototype. We believe that implementing a secure UI that allows the capabilities of commodity graphics hardware to be utilized will require additional hardware and software support. This is due to problems imposed by the massive complexity and resulting low assurance of today's video drivers. We discuss how to address these problems in 4.5 and 4.6 respectively.

3. PLATFORM SECURITY

3.1 Local Security Model

Terra's basic access control model is specified completely by the TVMM and the management VM. It is assumed that the management VM will make a distinction between the *platform owner* and *platform user*, similar to the distinction between system administrator and normal users in a standard OS access control model. We assume the platform owner can choose the TVMM (or OS) that boots, although only certain TVMMs will actually be trusted by third parties.

The trusted virtual machine monitor runs at the highest privilege level. It is "root secure," [54] meaning that it is secure from tampering even by the platform owner who has root level access, from the management VM, etc. The TVMM only dictates policy that is required for attestation; it isolates VMs from each other, it will not falsely attest to a VM's contents, and it will not disclose or allow tampering with the contents of a closed-box VM. The TVMM cannot guarantee availability. All other policy decisions are left to the discretion of the management VM, i.e. the platform owner.

The management VM formulates all platform access control and resource management policies. It grants access to peripherals, divides storage among VMs, and issues CPU and memory limits. It might formulate policies that limit how many VMs can run, which VMs can run (i.e. what software can run in a given VM), which VMs can access network interfaces or removable media, and so on. The management VM also starts, stops, and suspends VMs.

The management VM that runs is determined by the platform owner, so the security guarantees that the TVMM provides must not depend in any way on the management VM. The TVMM enforces these security guarantees, independent of the management

VM. The management VM does have the power to deny service to a VM, by failing to provide a required resource. This power is not a security failing because the platform owner and/or user can do the same thing, e.g. by unplugging the device.

3.2 Application Assurance

The most important property Terra provides for improving application security is allowing applications to determine their own level of assurance.

In traditional operating systems isolation between applications is extremely poor. The OS kernel itself has poor assurance and is easily compromised, and the great deal of state that is shared between applications makes it difficult to reason about isolation. As a result, compromising a single application often impacts a significant portion of the platform. Thus, the security of the entire platform is often reduced to that of its most vulnerable component. In Terra, applications in different VMs are strongly isolated from one another. This prevents the compromise of any single-application VM from impacting any other applications on the system. Thus, applications with greatly differing assurance requirements may run concurrently, because an application's level of assurance is independent of other applications on the system.

Terra's ability to run an application-specific operating system aids assurance in a variety of ways. Operating systems tailored to an application can be smaller and simpler than general-purpose OSes. Further, an OS tailored to an application can provide the best abstractions for satisfying the security requirements of that application. For example, the fine-grained access controls of SELinux [43] could be used to compartmentalize a more sophisticated application with many components, while a simpler application could reduce its TCB to simple bootstrapping code.

Attestation also has potential benefits for application assurance. Because applications can ensure that they only interact with trusted peers, they add an additional level of depth to their defenses. An attacker wishing to exploit the application must first either exploit its peer or find some means of impersonating its peer, in order to provide a vector for attack.

The assurance of applications running in Terra is still ultimately limited by the assurance of the operating system, in this case the TVMM. However, we believe that with adequate hardware support a VMM can provide isolation at the highest levels of assurance [3].

3.3 Trusting Software

Attestation allows a user to authenticate what hardware and software are in use on a remote platform. This is referred to as establishing that the remote platform is "trusted." This can be a valuable capability for gaining confidence in the integrity of a system's components, but correctly using this capability can also be quite subtle. In particular, it is easy to construe an attestation as promising more than it actually can. It is critical that developers not overestimate what this capability provides.

Naively assuming a client is completely trustworthy based on attestation can potentially make applications needlessly fragile and ultimately degrade their security instead of improving it. This can occur when a remote party places too much faith in the client's good behavior, ignoring relevant issues in the threat model, and making overly strong assumptions about software assurance or hardware tamper resistance. This can also occur if the trust provided by attestation is used as a replacement for stronger alternatives such as cryptography.

Attestation cannot make a promise about the future. A trusted node can fail at any time, through many means—hardware failure, power failure, a user unplugging the device—so an attestation cannot reliably guarantee that the node will do anything at a later time. At best, a trusted platform can only ensure the integrity and confidentiality of the software it is running. In this respect it is no worse than a real closed-box platform.

Hardware in the hands of malicious remote users can only be trusted up to the level of hardware tamper resistance. Current inexpensive commodity hardware that offers only modest tamper resistance, such as the minimal amount required to support TCPA [58], should generally be assumed to deter only the least resourceful attacker. It is our hope that as tamper-resistant hardware becomes more widely deployed, in the form of TCPA, high-quality tamper-resistant hardware will become affordable due to economy of scale. Effective means to take hardware tamper resistance and the threat from the local user and physical security into account in the design of trusted systems has been studied extensively. A good starting point on this topic is work by Anderson [6] and Yee [59].

4. TRUSTED VIRTUAL MACHINE MONITORS

Trusted virtual machine monitors provide application developers with the semantics of real closed-box platforms. VMs provide a raw hardware interface equipped with virtual network cards, video cards, secure disks, etc. VMs can attest to their contents by obtaining signed certificates using a direct interface to the TVMM. The TVMM provides the management VM with interfaces to create and manage VMs, and to connect them through virtual devices. In this section we describe these interfaces, and how they are implemented by the TVMM. We also describe hardware required for building TVMMs.

4.1 Storage Interface

The TVMM provides an interface for maintaining application security in the face of the threat model presented by mainstream tamper-resistant hardware such as a TCPA-equipped PC. It assumes that the hardware platform will provide tamper resistance for the memory, CPU, etc., but will not protect the disk. Thus, the disk may be removed from the machine, accessed by a different OS, etc. In light of this threat, Terra provides several classes of virtual disks that VMs can use to secure the privacy and integrity of their data. VMs can select what type of disk they are using for any given virtual disk, based on their security, performance, and functionality requirements:

Encrypted disks hold confidential data. The TVMM transparently encrypts/decrypts and HMACs [10] storage owned by a given VM on that VM's behalf, ensuring the storage's privacy and integrity.

Integrity-checked disks store mutable data whose integrity is important but does not require privacy. The TVMM uses a simple HMAC to prevent tampering. Optionally, a secure counter can prevent rollback (see section 4.6).

Raw disks provide unchecked storage. These are useful for sharing data with applications outside the VM.

In addition to these basic disk types, disks are also specified as being attested or unattested. Attested disks contain the program binary and other immutable state that make up the identity of the VM for the purpose of attestation. Which disks are attested is specified as part of a VM's metadata i.e. its basic configuration data. Persistent state that will change, e.g. variable configuration state or application data, is not kept on attestable disks, because a hash of its contents would not generally be meaningful to a remote party. Attestable disks may be encrypted or left in the clear at the discretion of the VM's developer.

Any VM that desires attestation must have been booted from an attestable disk. This disk's hash makes up the *primary identity* of the VM, along with the VM firmware and other immutable VM state. Additional disks may also be made attestable. The hash of each of these disks, if any, constitutes a secondary identity for the VM. The reason for this separation is to facilitate the specialization and redistribution of closed-box VMs. For example, suppose Acme firewall company produces a closed-box VM that provides trusted access point functionality (see section 5.3). The primary identity of this VM will be given by the VM supplied by Acme. Each company that purchases this box will add a separate disk which stores site-specific configuration data (e.g. firewall rules, VPN keys). The hash of this disk forms a *secondary identity* for the VM. A VM may have only one primary identity, but it may have any number of secondary identities.

Cryptographic keys used for protecting storage are sealed under the TVMM's public key (see section 4.6 for information on sealed storage). The hardware will release the TVMM's private key only to the TVMM itself, maintaining the confidentiality of these keys.

4.2 Implementing Attestation

In principle, computing the identity of an application for attestation is done by applying a secure hash to the entire executable image of an application before that application is started. In practice many issues must be taken into account. What portions of the VM are hashed? How is the VM decomposed for hashing? When are the hashes actually computed? The answers to all of these questions have important practical implications for security and performance.

A complete VM image consists of a variety of mutable and immutable data. The VM is defined not only by the initial contents of its virtual disks, but also by its NVRAM, system BIOS, PROMs for any BIOS extensions, and so on. Each VM also includes a "descriptor" that lists hashes for attestable parts of the VM, including attestable disks. The TVMM takes responsibility to ensure that loaded data actually matches these hashes.

Verifying an entire entity (e.g. a virtual disk) with a single hash is efficient only if the entity is always processed in its entirety. If subsections of a hashed entity are to be verified independently (e.g. demand paging a disk) then using a single hash is undesirable. So, instead of a single hash, Terra divides attestable entities into fixed-size blocks, each of which is hashed separately. The VM descriptor contains a hash over these hashes.

If the VM is accompanied by a list of the individual block hashes, subsections of the hashed entity can then be verified at a block-sized granularity, e.g. blocks can be verified as they are paged off disk. Whether the list of hashes is available or not, the entity as a whole can still be verified against the hash of hashes.

The problem of efficiency in hashing an entire entity is recursive. Hashing a 4 GB entity into 20-byte SHA-1 hashes with a 4 kB block size yields 20 MB of hashes. Storing these hashes on disk should not be a problem, since normal filesystems have a small per-block overhead anyway. A possible real problem is memory and time; before any of these hashes is used to verify a block, the entire 20 MB of hashes must themselves be verified against the hash in the VM descriptor. If it is too expensive to verify these 20 MB of hashes at startup or to keep them in memory, use of a Merkle hash tree [46] would trade startup delay for runtime performance. In the current Terra prototype we have not not yet implemented generalized hash trees to verify hashes, because we have not yet encountered space or performance constraints that necessitate their use.

Ahead-of-Time Attestation

Each stage in the boot process is responsible for signing a hash of the next stage before invoking it. All of these stages deal with small amounts of data that are loaded into memory in a single step. Thus, they are hashed in their entirety before they are given control. We call this "ahead-of-time attestation" because the attestation occurs before the code runs.

After boot, ahead-of-time attestation is appropriate for use with small, high-assurance VMs. The TVMM reads in the entire VM, verifies all of its attestable components against the VM's descriptor. It also pins the VM into physical memory to avoid the possibility of corruption due to malicious tampering.

Optimistic Attestation

Ahead-of-time attestation is impractical for larger VMs. The data to be verified must be both read and hashed. Both of these steps can take a significant amount of time. For example, ignoring disk transfer time, hashing 1 GB of data with OpenSSL's SHA-1 implementation takes over 8 seconds on a 2.4 GHz Pentium 4. (Section 5.2 measures performance of attestation in a real VM.) Moreover, any part of an attestable disk that is paged out and later read back must be verified again to detect malicious tampering.

To address these issues, we introduce the technique of "optimistic attestation." With optimistic attestation, the TVMM attests to whatever hashes the VM descriptor claims for its attestable disks, but it does not verify them at startup. Instead, individual blocks of the VM are lazily checked by the TVMM as they are read from disk at runtime. If a block fails to verify at the time it is read from disk, the TVMM halts the VM immediately.

Ahead-of-Time vs. Optimistic Behavior

Ahead-of-time attestation and optimistic attestation exhibit potentially different semantics. If attestation is done in advance, a single corrupted bit in an attestable disk prevents a VM from loading, but if attestation is performed optimistically, the VM will start and run until the first access to the corrupted block. VM designers may take this into account, but they should be aware that many kinds of events, including hardware failures and power outages, can cause a VM to stop suddenly at any time.

4.3 Attestation Interface

The TVMM provides a narrow interface to closed-box VMs for supporting attestation. This interface provides the following operations:

cert $\leftarrow$ ENDORSE(*cert-req*)

Places the VM's hash in the common name field of a certificate and places the contents of *cert-req* in the certificate. Signs the certificate with the TVMM's private key, and returns it to the VM. The *cert-req* argument contains the VM's public key and any other application data used for authenticating the VM. This function forms the basis of attestation.

hash $\leftarrow$ GET-ID()

Retrieves the hash of the calling VM. (The VM image cannot contain its own hash.) Useful for a VM that wishes to check whether the hash in an attestation from a remote party matches its own hash. This is frequently useful as closed boxes often have peers of the same type, e.g. the online game example shown in section 5.2.

4.4 Management Interface

Terra delegates VM administration duties to a special VM called the management VM. The management VM is responsible for managing the platform's resources on behalf of the platform owner, providing a user interface for starting, stopping, and controlling the execution of VMs, and connecting VMs through virtual device interfaces. The TVMM provides only basic VM abstractions. This simplifies its design as well as providing flexibility as policy can be completely determined by the management VM.

The TVMM provides basic services, such as support for running multiple, isolated VMs concurrently, but the management VM is responsible for higher-level resource allocation and management. In particular, the management VM allocates memory and disk space to VMs, and controls VM access to physical and virtual devices. It uses a function call interface into the TVMM to accomplish its tasks. The most important of these functions are outlined below:

device-id ← CREATE-DEVICE(*type*, *params*)

Creates a new virtual device of a given type with specified parameters, and yields a handle for the new device. The *type* may specify a virtual network interface, a virtual disk, etc. In the case of a virtual disk, *params* is a list of physical disk extents corresponding to the virtual disk's content. Other types of devices require other kinds of additional parameters.

CONNECT(*device-id-1*, *device-id-2*)
DISCONNECT(*device-id-1*, *device-id-2*)

Connects (or disconnects) the specified pair of devices. Each *device-id* is a virtual device id returned from CREATE-DEVICE or the well-known id of a physical device. When a pair of devices is connected, data output from one of them becomes input on the other and vice versa. For example, a virtual network device can be used to read and write network frames on a real network if it is connected to a physical network device.

vm-id ← CREATE-VM(*config*)

Prepares a VM to be run, and produces a handle for it. The parameter is a set of configuration attributes for the new VM. The configuration includes a pointer to the VM's descriptor. The VM by default has no attached devices.

ATTACH(*vm-id*, *device-id*)
DETACH(*vm-id*, *device-id*)

Attaches a given physical or virtual device to a VM, or removes one, respectively.

ON(*vm-id*)
OFF(*vm-id*)

Powers a VM up or down, respectively.

SUSPEND(*vm-id*)
RESUME(*vm-id*)

Temporarily prevents a VM from running or allows it to resume, respectively. The VM must already be on. (Individual VMs may disable this function.)

4.5 Device Driver Security

Device drivers pose an important challenge to TVMM security. Most of today's commodity platforms support a huge range of devices. Today's drivers can be very large (e.g. those for high-end video cards, software modems, and wireless cards) which makes gaining a high degree of confidence in their correctness virtually impossible. Further, there are a huge number of device drivers, and drivers frequently change to support new hardware features. Often these are written by relatively unskilled programmers, which makes their quality highly suspect. Empirically, driver code tends to be the worst quality code found in most kernels [16] as well as the greatest source of security bugs [8]. Given these facts, we cannot expect to include device drivers as part of the TVMM's trusted computing base.

The problem of untrusted device drivers has currently not been addressed by our TVMM prototype. However, a variety of solutions exist. Protecting the TVMM from untrusted device drivers requires several problems to be addressed. First, the TVMM must be protected from direct tampering by the driver code. This is achieved by confining drivers via hardware memory protection and restricting their access to sensitive interfaces. A wide variety of systems have addressed this problem, from exotic microkernel [42, 45] and safe language based systems [11] to practical adaptations to existing operating systems, such as Nooks, which provides device driver isolation for fault tolerance in Linux [56].

A further threat that must be addressed is posed by malicious devices using hardware I/O capabilities (e.g. hardware DMA) to modify the kernel. Addressing this requires additional assistance from the I/O MMU or similar chip set. One approach has been demonstrated in a modified version of the Mungi system [40], that runs device drivers at user level, as independent processes, and prevents them from performing DMA outside their own address spaces.

Another approach to this problem is specified by the forthcoming NGSCB architecture. In NGSCB the issue of supporting device drivers is avoided altogether by leveraging the device drivers of an untrusted operating system (e.g. Windows XP) that runs concurrently on the platform. In NGSCB, a trusted operating system such as a TVMM can run in "curtained memory," memory that is protected from tampering by both the untrusted operating system, and from "attacks from below" via DMA. The trusted operating system leverages the device drivers of the untrusted operating system by interfacing with them via an explicit interface in the untrusted OS's kernel. As a side benefit of this approach, the TVMM does not need to provide its own drivers and instead can leverage those of an existing operating system (e.g. Windows). Leveraging the drivers of another operating system to support a TVMM would be very similar to the hosted VMM approach of VMware Workstation [55].

Untrusted device drivers pose another problem. If the TVMM cannot trust device drivers, it cannot rely on them to provide a trusted path. Overcoming this challenge requires additional hardware support, discussed below.

4.6 Hardware Support for Trusted VMMs

Terra relies on the presence of a variety of hardware assistance:

Hardware Attestation Minimally, the hardware must be able to attest to the booted operating system.

Sealed Storage Encrypts data under the private key of the tamper-resistant coprocessor that is responsible for attestation etc. (e.g. a TPM in the TCPA architecture). A hash of the booted trusted OS is also included with the encrypted data. The coprocessor will only allow a trusted OS with the same hash that sealed data to unseal it. This functionality is used by the TVMM to store its private key on persistent storage. Using this functionality ensures that hardware will only release a TVMM's private key to it to the same TVMM that stored it.

Both of these features are currently supported by TCPA. Several other forms of hardware support are desirable:

Hardware Support for Virtualization Specialized hardware support for accelerating virtualization has long been available in IBM mainframes [33]. We believe this type of hardware support significantly eases the burden of implementing a virtual machine monitor capable of efficiently handling the operating system diversity of commodity computing platforms. Hardware assistance is especially important for efficient interfacing to complex hardware such as graphics and 3-D accelerators. Additional hardware support can also greatly simplify virtualization, allowing very simple VMMs to be built, which in turn aids security.

Hardware Support for Secure I/O As discussed above, we cannot assume trust in device drivers on commodity platforms. Given this, it is essential to provide some means of establishing a secure connection between the TVMM and devices required to provide a trusted path (e.g. mouse, keyboard, video card, etc.). One way to accomplish this is by use of cryptography to secure communication between hardware devices and the TVMM. This could be supported either through additional support for encryption on new devices, or by way of hardware dongles to support legacy devices. Clearly encrypting all communication with the device would simply necessitate moving the driver into the TVMM. Thus, another step to supporting secure I/O would be splitting device interfaces. For example, on video cards the interface could be split into a simple 2D interface that could run in the TVMM and be used to implement the secure UI. A sophisticated 3D interface could be exposed directly to VMs, enabling high-performance graphics operations.

Secure Counter A secure counter, that is, a counter that can only be incremented, greatly enhances the functionality of a VM [23]. A secure counter is necessary to guarantee freshness, e.g. to prevent filesystem rollback attacks. A secure real-time clock is also useful, e.g. for expiring old session keys, defending against replay attacks, and rate limiting (discussed in section 5). Secure clocks are currently difficult to manufacture inexpensively, so for now it may be necessary to make do with secure counters.

Device Isolation The TVMM would like to protect itself and the VMs it runs from attacks from below, i.e. attacks coming from devices that have access to the DMA controller, PCI bus, etc. Hardware support for controlling access to these resources, in particular to shield VMs and the TVMM from attack would greatly increase the platform's security, because we would not have to trust device drivers. We anticipate that support for limiting device access to DMA, etc., will soon be present in commodity PCs to support Microsoft's NGSCB architecture [13, 4].

Real-Time Support Closed-box applications often have real-time requirements (e.g. game consoles, cellular phones) that cannot be satisfied by today's operating systems or VMMs. We believe additional hardware support could aid in addressing this problem as well. How best to accommodate these through a combination of low-level virtualization techniques and resource management is a topic for future work.

5. EXPERIENCE AND APPLICATIONS

In this section we describe the Terra prototype and provide an in-depth discussion of several applications that we built using the prototype. We also look at how these applications demonstrate the capabilities and the limitations of the closed-box abstraction that Terra provides. We also discuss other potential applications.

5.1 Prototype Implementation

We built a prototype of the trusted virtual machine monitor using VMware GSX Server 2.0.1 with Debian GNU/Linux as the host operating system. Neither Debian nor VMware GSX Server is suitably high assurance for a real TVMM, but they form a convenient platform for experimentation. In practice the same techniques that we describe here can be applied to a dedicated VMM offering high performance [34] and assurance, such as a hypothetical lightweight client-side version of VMware ESX Server [60].

Communication between VMs and the TVMM's attestation device is implemented with a VMware virtual serial device. A Python program monitors the host end of this device and handles requests specified by the attestation interface (section 4.3).

We currently do not attempt to emulate the underlying TCPA hardware that the TVMM would communicate with. We believe that since these interactions are relatively minimal and well understood, adding it to our prototype system would be superfluous.

Secure Storage

To implement optimistic attestation and other changes to the way VMware GSX Server uses storage, we had to modify the way it accesses virtual disks. We achieved this by interposing on the VMM's read and write operations using a dynamic preload library. This allowed us to modify the underlying implementation of virtual disks to support our new disk types without the need to change the VMM's source code, which was not available to us.

Ahead-of-time attestation was implemented by verifying whole file hashes before a VM is started. For optimistic attestation, our shared library verifies hashes as data is read from the files that VMware GSX Server uses to represent a virtual disk. Only aligned, full-size blocks can be verified with hashes, so the preload library extends the start and end positions of each read to the edge of an aligned block boundary. Misaligned or partial block writes also require one or two block reads. The same strategies are applied to accesses to integrity-checked and encrypted storage. We use bounce buffers to prevent the VM from seeing unverified data, although for performance a real implementation might try to avoid them on aligned full-block reads, perhaps by temporarily marking pages inaccessible.

System Management

A Python program implements the management VM utilizing the interface described in section 4.4. It currently only provides a simple means of managing VMs for testing purposes. The management interface is a Python wrapper layered over a variety of management and configuration interfaces provided by VMware GSX Server.

For certificate management we relied on the OpenSSL library. Our certificates are in X.509v3 format, with X.509 certificate requests used to request attestation. Currently the "common name" field is used for the attestation hash; an extension field would be more suitable. The prototype uses a single trusted CA, which signs a hardware certificate, which signs the TVMM's certificate. The TVMM in turn signs each application's attestation certificate.

5.2 Trusted Quake

Commercial multiplayer online games have soared in popularity since the mid-1990s. As the popularity of these games has increased, so has the incidence of cheating. Cheating in these games most often occurs when a malicious party alters the client, the server, or their data files to unfairly change the rules of the game. Cheaters may also take advantage of insecure communication between clients and servers, either to spy on their opponents or maliciously alter traffic.

To better understand how to combat these problems in a real-world online game, we built "Trusted Quake," a closed-box version of "Quake II" [35], a popular "first-person shooter" with a long and storied history of problems due to cheating.

Trusted Quake runs Quake in a closed-box VM and uses attestation to ensure that all of the hosts it contacts, whether clients or servers, also run the same version of Trusted Quake. The attestation protocol is used to exchange 160-bit SHA-1 HMAC keys [10] and 56-bit DES keys. All normal Quake traffic is then exchanged using the HMAC and DES keys for integrity and confidentiality, respectively. The TVMM will not falsely attest that a different VM is Trusted Quake, and the isolation properties of the TVMM keep the keys from leaking.

Our prototype of trusted Quake uses a VM running a minimal Linux 2.4.20 kernel on top of a minimal installation of Debian GNU/Linux 3.0. The VM boots directly into Quake. No shell or configuration interface is available to users. A dynamic preload library interposes on Quake's network communication to perform attestation and key exchange. It uses a custom user-space implementation of the IPsec Encapsulating Security Payload (ESP) protocol [2] to provide both integrity and confidentiality. DNS traffic is special-cased, with the preload library checking incoming DNS responses for proper formatting to allow interaction with conventional DNS servers.

We measured the time for the Trusted Quake VM to boot with different forms of attestation. Booting without any form of attestation takes 26.6 seconds. Ahead-of-time attestation adds 30.5 seconds, totaling 57.1 seconds. Substituting optimistic attestation, boot totals only 27.3 seconds. Adding encryption to optimistic attestation raises the total boot time to 29.1 seconds. (Times are averaged over five runs.) We conclude that optimistic attestation has significant benefits for VM startup. As for interactive performance after boot, we found it to be subjectively indistinguishable from untrusted Quake running within a VM.

Security in Quake, as in many such games, originally took the form of "security by obscurity." However, given its huge popularity it was not long before its binary, graphics and audio media files, and network protocol were reverse-engineered by those intent on modifying the game. These modifications led to development of a wide variety of well-documented ways to cheat, by observing and modifying the game client, server, and network traffic.

The security properties provided by Trusted Quake prevent many common types of cheating and other security problems in untrusted Quake:

Secure Communication The secrecy provided by the closed-box VM allows Quake to maintain a shared secret that it can use to securely communicate with its peers. This defeats several forms of cheating. First, since Trusted Quake authenticates all of its traffic, traffic cannot be forged from it, nor can its traffic be modified. This defeats active attacks in the form of aiming proxies [19], agents that interpose on game traffic on behalf of a player to improve aiming. It also defeats passive attacks. When users can observe opponents' Quake network traffic, they can find out important information about game state, such as the location of other players.

Client Integrity Edited client 3D models can facilitate cheating, e.g. modified models of opposing players can make them visible from farther away or around corners. Similarly, clients can modify sounds that indicate nearby players, making them louder or more distinctive [26]. Some Quake variants verify weak checksums of models to attempt to prevent this type of cheating, but these can be bypassed using modified clients or modified models that still match the expected checksum [26]. Trusted Quake frustrates these attacks because users cannot edit files in the Trusted Quake VM.

Server Integrity The Quake server coordinates and controls the game. It is often run by one of the players in a game, so incentive to cheat is strong. A trusted server prevents two kinds of problems. First, it prevents cheating by the server itself, in which the server offers advantages to selected players. Second, it allows only trusted clients to connect, preventing cheating by individual players.

Isolation A corollary of isolating Quake is that the rest of the system is protected if Quake is misbehaving due to remote compromise.

Trusted Quake cannot prevent some kinds of cheating:

Bugs and Undesirable Features Quake has some commands that inadvertently allow cheating. For instance, one command displays the number of rendered polygon models on-screen. When this number increases, it can indicate that another player is about to come into view. Another command can be used to simulate network lag, allowing the player to hang in mid-air for a limited time.

Network Denial-of-Service Attacks Trusted Quake does not affect attacks that prevent communication between a client and a server. This can be used to introduce lag into other players' connections, putting them at a disadvantage. This is especially easy for the server's owner, who has direct control over outgoing packets.

Out-of-Band Collusion Multiple players who are physically near each other can gain extra information by watching each others' monitors or talking to one another, which may allow them an unfair advantage over opponents. Similar cheating is possible via telephone or online chat services.

Trusted Quake provides a specific example of a solution to the very general problem of protecting the privacy and integrity of a complex service in the face of a variety of threats. The techniques we applied here could be used to improve the security of a wide variety of online games, as well as other types of multi-user applications. Trusted Quake also illustrates the limitations of this technique. Even given the features that Terra provides, it is no panacea. Applications must still be carefully designed and some forms of attack simply cannot be prevented with the features Terra provides.

5.3 Trusted Access Points (TAPs)

Trusted Quake illustrates how a specific application can be hardened to ensure that it acts as a well-behaved peer. However, for many applications it is not necessary to harden the entire application. Rather, we simply want to ensure that its communication is well regulated, e.g. rate limited, monitored, access controlled. This can be achieved with a trusted access point (TAP), that is, a filter for network traffic that runs on each client that wishes to access the network. The TAP examines both incoming and outgoing packets and forwards only those that conform to policy. A TAP system can be used to secure the endpoints of overlay networks such as corporate VPNs, to secure point-to-point connections to access points such as wireless APs, dial-in access, and even standard wired gateways [28], or simply to regulate access to network service.

We implemented a TAP system designed to allow a company (or other entity) to securely grant outside visitors limited access to its internal network. To receive this limited use of the internal network via the TAP system, a machine's owner physically connects the machine to the "restricted network," that is, a network isolated from the internal network, then installs the TAP closed-box VM on it. This VM contains a VPN client and firewall software for filtering packets. At startup, the VPN client connects to a TAP gateway that bridges the internal network to the restricted network. The client attests itself to the TAP gateway, and the client and gateway server exchange secret parameters used for encrypting and integrity-checking data packets between the two machines.

The TAP VM can implement a traditional network policy preventing IP spoofing, unapproved port usage, rate-limiting, etc. Only packets that adhere to policy are permitted to pass between the internal and restricted networks. The TAP VM can also implement more complex network policy, running remote vulnerability scanners like Nessus and network intrusion detection systems like Snort. When applied to large numbers of clients by a single server, these can consume considerable network and computational resources. Pushing these costs to the client significantly eases the burden.

As with our Quake VM, the TAP prototype runs a minimal Linux 2.4.20 kernel sitting on top of a minimal Debian 3.0 installation within a closed-box VM. The VM is single-purpose and has no user interface. We use the popular OpenVPN secure IP tunnel daemon, version 1.3.0, to transmit packets between the TAP VM and the TAP gateway. Key exchange and certificate presentation is carried out over SSL, a built-in feature of OpenVPN. On the TAP gateway we check the client's certificate via OpenVPN's ability to do so using an external program.

Benefits

Use of a TAP system has several benefits:

Prevents Source Forging The TAP VM can reject packets whose source address does not match the address assigned to the machine.

Prevents DoS Attacks The TAP VM can detect denial-of-service attacks on machines in the internal network and throttle service at the source. (Attempts at source forging might be a sign of a DoS attempt.) Self-detection of DoS attacks could be augmented by notification from an authority on the internal network.

Scalability A centralized router can be overloaded relatively easily if each packet must traverse an entire TCP/IP stack, go through a network intrusion detection system, and so on. When the client that wishes to send or receive packets is also responsible for verifying them, scalability is improved.

Network Scalability Vulnerability scans, such as port scans, can consume considerable network bandwidth. Performing scans between VMs within a computer, instead of over a wire, reduces bandwidth costs and may allow the frequency of scans to be increased.

TAP systems do have limitations. In particular, there can be no assumption that all packets on a wire are authenticated using a TAP system. Nothing prevents an untrusted host from physically connecting to the network, and nothing prevents a trusted host from rebooting into an untrusted OS or bypassing the TAP VM. Thus, attacks, such as flooding attacks, on the restricted network cannot be prevented. However, if individual ports on a switch can be limited to pass only properly HMAC'd or encrypted packets, with some provision for initial negotiation of keys, then this issue can be eliminated.

5.4 Additional Applications

We have explored just a few of the potential applications of this platform. It can support a wide range of other applications, including:

- High-Assurance Terminals

  Many applications require a trusted platform as a secure platform for sending or receiving relatively basic information from the user. In these situations we leverage three properties: the platform's ability to provide a trusted path to and from the user, its ability to support high-assurance applications that are highly robust in the face of a remote attacker, and the remote host's ability to ensure that the user is following best practice by running a closed-box version of the application.

  One example is "feeds" that report current stock prices, news, and other data that financial analysts use to make decisions. Such interfaces must be extremely reliable. Malicious manipulation of these applications could have devastating consequences for individual traders, whole firms, even entire financial markets. This capability could also be used to provide voting stations that attest their integrity to the remote tabulation service.

- Isolated Monitors

  The strong isolation provided by Terra's use of a VMM is by itself extremely useful. This can be used to harden a variety of host security mechanisms against attack, such as key stores, intrusion detection systems [27], secure logging systems [21], and virus scanners [4].

- Virtual Secure Coprocessors

  Many applications studied in the context of secure coprocessors such as the IBM 4758 [22, 53] also lend themselves to implementation in this architecture. Some of these applications include privacy-preserving databases [54, 36], secure auctions [49], and online commerce applications [63]. A key constraint in adapting applications from such architectures to a trusted platform like Terra will be ensuring that the platform provides an adequate level of hardware tamper resistance for the application.

Clearly, the range of specific applications that can benefit from the general mechanisms provided by Terra is far too long to list. More specific mechanisms that could leverage Terra such as desktop separation [47], application sandboxing, and OS authentication [62] have already been explored elsewhere.

6. RELATED WORK

The central mechanism in our work is the virtual machine monitor. Extensive discussion of VMMs and their properties is found in seminal work by Goldberg [32, 33] and more contemporary work on Disco [12] and VMware [55, 60]. More recently, Chen [15] argues for routine and extensive use of VMMs for security purposes.

Our primary reason for choosing a VMM based architecture is the flexibility it provides. Our claim is that a trusted operating system best serves developers by providing a hardware abstraction as a typical closed platform would, thereby providing maximum flexibility. A more general argument about the inherently limiting nature of committing to a single OS abstraction has been made by the extensible OS community, perhaps most concisely in arguing for exokernels [24]. Exokernels and VMMs are in many ways quite similar. They are primarily differentiated by the fact that an exokernel's resource abstractions are optimized for performance, whereas those of a VMM are optimized for compatibility.

Computer systems able to cryptographically demonstrate their security properties to other systems are mentioned first in the work on trusted computing systems [57] and security kernels [30, 50] from the late 1970s and early 1980s. These systems took the principle of least privilege to the extreme in a general-purpose operating system, relying on a small kernel to do isolation, while all other operating functions, such as memory management and process scheduling, were pushed upward into less-trusted code. It was found that this led to systems that by and large were extremely inefficient, for diminishing returns in simplicity. Reported experience with these systems, especially those to kernelize the already svelte VM370 [17] in the form of KVM370 [31, 51], led us to believe that the VMM represents a least common denominator for virtualization, simplification beyond which yields little additional benefit [30].

The concept of authenticating a platform's software stack was fully developed in the Distributed System Security Architecture of Gasser et al. [29]. This work had all the essential components found in today's architectures for trusted computing, such as TCPA [58]. Each computer system contained dedicated hardware with a public/private key pair that it could use to authenticate to others the identity of the system it had booted by signing a hash of the boot image. The operating system (VMS) could in turn use its own key pair to sign for applications loaded by the system, etc., allowing a system's software to fully authenticate itself to a remote system. Including the machine as part of the authentication process, explicitly taking its composition into account, was also included in the authentication systems developed in later work on Taos [62]. This approach is treated thoroughly by Lampson et al. in their related treatise on authentication in distributed systems [39]. The more recent IBM 4758 secure coprocessor [22, 53] also allows for authenticating the source of outbound connections. Authentication in Terra differs most prominently from this previous work on platform authentication in that an application's software stack is treated as a single authenticated unit, in contrast with previous solutions which authenticated to individual parts of an applications software stack in a piecemeal fashion. Terra's support for rapid authentication of large applications further distinguishes it from previous systems. On the opposite end of the spectrum, Execute Only Memory (XOM) [41] uses cryptographic hardware in the processor to preserve the privacy and integrity of code running in a process on an untrusted operating system. It provides much less functionality than Terra for building secure applications, such as a trusted path to I/O devices.

Efforts by Yee and Tygar on Dyad [59] explored hardware mechanisms to bootstrap trust in the host with secure coprocessors on standard PC hardware. More importantly, this work brought to light the practical applications of this technology for consumers, such as electronic currency, stamps, and copy protection, and articulated a vision of including such hardware on mainstream PCs. The AEGIS system by Arbaugh [7] provides a practical foundation for implementing secure boot on a PC. AEGIS uses a signed hash to identify each layer in the boot process, as does Terra. Unlike Terra, the primary purpose of AEGIS is to ensure that only a single authorized software stack can be loaded on a machine. Terra's signatures are designed to prove to third parties the software running on the machine, whereas those in AEGIS enforce booting only a single software stack.

Recently, hardware support for sealed storage and attested boot has become available in the form of commodity platforms implementing TCPA. TCPA 1.1b [58] provides all the basic features to support Terra, although the addition of some of the optional features described in section 4.6, such as improved support for device isolation, secure counters, etc., are certainly desirable, and may be forthcoming in the as-yet-unreleased TCPA 1.2 specification. TCPA is only a hardware mechanism for trusted computing, lacking a vision for support of trusted computing in operating systems.

In recognition of this need for OS support for trusted computing, Microsoft began development of its NGSCB (formerly Palladium) architecture [13, 4, 23, 48, 25]. This work is the most similar to ours in that it provides a "whole system" solution to the problem of trusted computing. NGSCB works by partitioning the platform into two parts ("trusted" and "untrusted") each of which runs a different operating system. It achieves this through what can be seen as a very special purpose VMM that only supports two VMs. The untrusted part runs one of today's commodity operating systems (e.g. Windows) while the trusted part runs a dedicated trusted operating system (the "nexus" in NGSCB parlance). This dedicated operating system is designed to run small, high-assurance programs called "agents." Agents work in conjunction with code on the untrusted side of the system, providing all of the security-critical functionality that programs on the untrusted side need (e.g. sensitive key storage).

NGSCB differs from Terra most prominently in its programming model and how it supports high-assurance applications. Terra allows application designers to specify any OS they desire for closed-box applications. In contrast, NGSCB requires application designers to target their closed-box applications to a single, specific Microsoft OS. Terra also differs in its attestation model. In Terra an application's entire software stack is attested, while in NGSCB only agents are attested. Superficially it appears that Terra provides a more flexible model for building applications, but making any concrete comparison at this point would be difficult, because the NGSCB software architecture is as yet largely unpublished.

Ultimately, NGSCB's architecture may complement Terra's. It appears that hardware support for NGSCB may be fairly OS neutral, thus allowing other architectures (such as Terra) to take advantage of the trusted path support in devices, hardware support for isolation, etc. that it provides. Likewise, an architecture like Terra that can provide an arbitrary number of compatible VMs should be able to host a software architecture like NGSCB which requires just two VMs.

We presented the initial idea of providing a closed-box abstraction for trusted computing through the use of a virtual machine monitor in a short position paper [28].

7. CONCLUSION

We presented a flexible architecture for trusted computing, called Terra. Terra allows applications to run in an "open box" VM with the semantics of a modern open platform, or in a "closed box" VM with those of dedicated, tamper-resistant hardware. The key primitive that Terra builds on is a trusted virtual machine monitor (TVMM). The TVMM mechanisms allow Terra to partition the platform into multiple, isolated VMs. Each VM can tailor its software stack to its security and compatibility requirements.

We examined the primitives the TVMM provides for building closed-box VMs, in particular those required to support "attestation," the mechanism used to cryptographically identify the contents of closed-box VMs to remote parties. We described how to efficiently implement these primitives. We implemented these primitives in a prototype implementation of Terra and built a selection of applications using this prototype that demonstrate its capabilities. We believe that the closed-box VM abstraction provided in the Terra architecture forms the basis for a truly general-purpose trusted computing platform.

8. ACKNOWLEDGMENTS

Our shepherd Ted Wobber gave us a great deal of valuable feedback that significantly improved the quality of this work. Discussions and feedback from Matthias Jacobson, Ulfar Erlingsson, and John DeTreville also greatly assisted us, as did the comments of our anonymous reviewers. Discussions with Paul England and Marcus Peinado were very valuable in formulating our ideas about trusted computing and gaining insight into the motivation behind the NGSCB design. We are very grateful to Cristen Torrey and Kathryn Waffle for their editorial assistance and moral support. This work was supported in part by the National Science Foundation under Grant No. 0121481, the Packard Foundation, and Stanford Graduate Fellowships.

9. REFERENCES

[1] IBM mainframe servers: Case studies. `http://www-1.ibm.com/servers/eserver/zseries/library/casestudies/`.

[2] IP security protocol (IPsec) charter. `http://www.ietf.org/html.charters/ipsec-charter.html`.

[3] Security: IBM zSeries partitioning achieves highest certification. `http://www-1.ibm.com/servers/eserver/zseries/security/certification.htm%l`, December 2002.

[4] Microsoft next-generation secure computing base—technical FAQ. `http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/%news/NGSCB.asp`, February 2003.

[5] R. Anderson. Cryptography and competition policy: Issues with trusted computing. In *Proc. Workshop on Economics and Info. Sec.*, pages 1–11, May 2003.

[6] R. Anderson and M. Kuhn. Tamper resistance—A cautionary note. In *Proc. 2nd USENIX Workshop on Electronic Commerce*, pages 1–11, Nov. 1996.

[7] W. Arbaugh, D. Farber, and J. Smith. A secure and reliable bootstrap architecture. In *Proc. 1997 IEEE Symp. Sec.*, pages 65–71, May 1997.

[8] K. Ashcraft and D. Engler. Using programmer-written compiler extensions to catch security holes. In *IEEE Symp. Sec. and Privacy*, Oakland, May 2002. IEEE, IEEE Computer Society Press.

[9] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Proc. CRYPTO'2000*, 2000.

[10] M. Bellare, R. Canetti, and H. Krawczyk. Message authentication using hash functions—the HMAC construction. *CryptoBytes*, 2(1), Spring 1996.

[11] B. Bershad, S. Savage, P. Pardyak, E. Sirer, M. Fiuczynski, D. Becker, C. Chambers, and S. Eggers. Extensibility, safety, and performance in the SPIN operating system. In *Proc. 15th ACM Symp. on Operating Sys. Principles*, Dec. 1995.

[12] E. Bugnion, S. Devine, and M. Rosenblum. Disco: running commodity operating systems on scalable multiprocessors. In *Proc. 16th ACM Symp. Operating Sys. Principles*, Oct. 1997.

[13] A. Carroll, M. Juarez, J. Polk, and T. Leininger. Microsoft Palladium: A business overview. `http://www.microsoft.com/PressPass/features/2002/jul02/0724palladiumwp.asp`, August 2002.

[14] D. Chaum and E. V. Heyst. Group signatures. *Advances in Cryptology, Eurocrypt '91*, 547:257–265, 1991. Springer-Verlag Lecture Notes on Computer Science.

[15] P. M. Chen and B. D. Noble. When virtual is better than real. In *Proc. 2001 Workshop on Hot Topics in Operating Sys. (HotOS-VIII)*, Schloss Elmau, Germany, May 2001.

[16] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. An empirical study of operating system errors. In *Proc. 18th ACM Symp. Operating Sys. Principles*, Oct. 2001.

[17] R. J. Creasy. The origin of the VM/370 time-sharing system. *IBM J. Research and Development*, 25(5):483–490, September 1981.

[18] P. Cummings, D. Fullan, M. Goldstien, M. Gosse, J. Picciotto, J. Woodward, and J. Wynn. Compartmented model workstation: Results through prototyping. In *Proc. IEEE Symp. Sec. and Privacy*, pages 27 – 29, April 1987.

[19] DarkNova. Interview with an aimbot coder. `http://www.lamerkatz.com/webvoid/issue7/1.shtml`.

[20] J. J. Donovan and S. E. Madnick. Hierarchical approach to computer system integrity. *IBM Sys. J.*, 14(2):188–202, 1975.

[21] G. W. Dunlap, S. T. King, S. Cinar, M. Basrai, and P. M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proc. 2002 Symp. Operating Sys. Design and Implementation*, December 2002.

[22] J. Dyer, M. Lindemann, R. Perez, R. Sailer, S. Smith, L. van Doorn, and S. Weingart. Building the IBM 4758 secure coprocessor. *IEEE Comp.*, 34:57–66, October 2001.

[23] P. England. Personal communication.

[24] D. Engler, M. Kaashoek, and J. O'Toole. Exokernel: An operating system architecture for application-level resource managment. In *Proc. 15th ACM Symp. on Operating Sys. Principles*, Dec. 1995.

[25] P. Englund, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman. A trusted open platform. *IEEE Spectrum*, pages 55–62, 2003.

[26] Flocutus. The ultimative Quake cheating page: Illegitimate cheats. `http://www.gamescenter.de/uqc/illegal.htm`.

[27] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. Net. and Distributed Sys. Sec. Symp.*, February 2003.

[28] T. Garfinkel, M. Rosenblum, and D. Boneh. A Broader Vision for Trusted Computing. In *9th Workshop on Hot Topics in Operating Sys. (HotOS-IX)*, May 2003.

[29] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The digital distributed system security architecture. In *Proc. 12th NIST-NCSC Nat'l Comp. Sec. Conf.*, pages 305–319, 1989.

[30] B. Gold, R. Linde, and P. Cudney. KVM/370 in retrospect. In *Proc. IEEE Symp. Security and Privacy*, April 1984.

[31] B. Gold, R. Linde, R. J. Peller, M. Schaefer, J. Scheid, and P. D. Ward. A security retrofit for VM/370. In *AFIPS Natl. Comp. Conf.*, volume 48, pages 335–344, June 1979.

[32] R. Goldberg. *Architectural Principles for Virtual Computer Systems*. PhD thesis, Harvard University, 1972.

[33] R. Goldberg. Survey of virtual machine research. *IEEE Computer Magazine*, 7:34–45, June 1974.

[34] R. Howworth. Virtual servers pay off. IT Week, March 2003.

[35] I. id Software. Quake. `http://www.idsoftware.com/games/quake/`.

[36] A. Iliev and S. Smith. Prototyping an armored data vault: Rights management on Big Brother's computer. *Privacy-Enhancing Technology*, 2002. Springer-Verlag Lecture Notes on Computer Science.

[37] G. Jain. Certificate revocation: A survey. `http://www.cis.upenn.edu/˜jaing/papers/`.

[38] P. Karger, M. Zurko, D. Bonin, A. Mason, and C. Kahn. A retrospective on the VAX VMM security kernel. In *IEEE Trans. Soft. Eng.*, Nov. 1991.

[39] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice.

ACM Trans. Comp. Sys., 10(4):265–310, 1992.

[40] B. Leslie and G. Heiser. Towards untrusted device drivers. Technical Report 0303, University of New South Whales, March 2003.

[41] D. Lie, C. A. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. C. Mitchell, and M. Horowitz. Architectural support for copy and tamper resistant software. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, pages 168–177, 2000.

[42] J. Liedtke. On μ-kernel construction. In *Proc. 15th Symp. on Operating Sys. Principles*, pages 237–250, December 1995.

[43] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In *Proc. USENIX Tech. Conf., FREENIX Track*, pages 29–42, 2001.

[44] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell. The inevitability of failure: The flawed assumption of security in modern computing environments. In *Proc. Nat'l Info. Sys. Sec. Conf.*, pages 303–314, October 1998.

[45] M. Accetta et al. Mach: A new kernel foundation for UNIX development. In *Proc. USENIX Summer Conf.*, 1986.

[46] R. Merkle. Protocols for public key cryptosystems. In *IEEE Symp. Security and Privacy*, Oakland, April 1980. IEEE, IEEE Computer Society Press.

[47] R. Meushaw and D. Simard. NetTop: Commercial technology in high assurance applications. `http://www.vmware.com/pdf/TechTrendNotes.pdf`, 2000.

[48] paul england and marcus Peinado. Authenticated operation of open computing devices. In *Proc. 7th Australian Conf. Info. Sec. and Privacy*, pages 346–361, 2002. Springer-Verlag Lecture Notes on Computer Science.

[49] A. Perrig, S. Smith, D. Song, and J. Tygar. SAM: A flexible and secure auction architecture using trusted hardware. *eJETA.org: The Electronic Journal for E-Commerce Tools and Applications*, 1(1), January 2002.

[50] S. R. Ames, Jr. Security kernels: A solution or a problem? In *Proc. IEEE Symp. Sec. and Privacy*, April 1981.

[51] M. Schaefer and B. Gold. Program confinement in KVM/370. In *Proc. 1977 Ann. ACM Conf.*, pages 404–410, October 1977.

[52] J. S. Shapiro, J. M. Smith, and D. J. Farber. EROS: A fast capability system. In *Symp. on Operating Sys. Principles*, pages 170–185, 1999.

[53] S. W. Smith. Outbound authentication for programmable secure coprocessors. In D. Gollman et al., editor, *ESORICS 2002: 7th European Symp. Research in Comp. Sec.*, volume 2502/2002, pages 72–89, Zurich, Switzerland, October 2002. Springer-Verlag Heidelberg.

[54] S. W. Smith and D. Safford. Practical server privacy with secure coprocessors. *IBM Sys. J.*, 40(3):683–695, 2001.

[55] J. Sugerman, G. Venkitachalam, and B. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proc. 2001 Ann. USENIX Tech. Conf.*, Boston, MA, USA, June 2001.

[56] M. M. Swift, B. N. Bershad, and H. M. Levy. Improving the reliability of commodity operating systems. In *Proc. 19th Symp. on Operating Sys. Principles*, October 2003.

[57] P. S. Tasker. Trusted computer systems. In *Proc. IEEE Symp. Sec. and Privacy*, April 1981.

[58] Trusted Computing Platform Alliance. TCPA main specification v. 1.1b. `http://www.trustedcomputing.org/`.

[59] J. D. Tygar and B. Yee. Dyad: A system for using physically secure coprocessors. In *IP Workshop Proc.*, 1994.

[60] C. A. Waldspurger. Memory resource management in VMware ESX Server. In *Proc. 2002 Symp. Operating Sys. Design and Implementation*, December 2002.

[61] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and performance in the Denali isolation kernel. In *Proc. 5th USENIX Symp. on Operating Sys. Design and Implementation*, December 2002.

[62] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. *ACM Trans. Comp. Sys.*, 12(1):3–32, 1994.

[63] B. Yee and D. Tygar. Secure coprocessors in electronic commerce applications. In *Proc. 1st USENIX Workshop on Elec. Commerce*, New York, New York, July 1995.