Abstract. Nowadays, more and more companies tend to use virtual machines instead of physically separated machines as platform for their IT services. This reduces the hardware costs and also simplifies the management of the IT infrastructure. However, virtualized environments do, in contrast to popular belief, not necessarily increase the security of a system. Actually, virtualized environments face the same threats as non-virtualized environments. In addition, there are considerably fewer security solutions for virtualized systems than for non-virtualized environments. To solve this problem, researchers have proposed the use of so-called secure hypervisors in order to provide a Trusted Computing Base for virtualized systems. This paper will present and discuss three of these secure hypervisor approaches in order to give the reader a better understanding of what secure hypervisors are and how they can improve the security of virtualized systems. Furthermore, it will compare the concepts behind the three approaches to show the actual state of affairs of secure hypervisor research. The three secure hypervisor approaches that were selected for this purpose are Terra, the VAX VMM, and sHype.

1 Introduction

In recent times, information security has become more and more important for all of us. While some years ago only government bodies, security experts, and system administrators were concerned about digital threats, nowadays not a single day passes by without a newspaper or a magazine reporting about a security incident or a newly discovered threat. The reason for this change of thought has a lot to do with advances in technology. Today, information technology is all around us and with technological progress a lot of the threats that existed in the physical world have shifted to the digital world. Consider fraud for example. In the past criminals sent letters to their victims, called them on the phone, or had to approach them personally to lure them into buying something worthless or nonexistent. These days there are emails, chats, and websites. Almost every computer is connected to some sort of network and therefore digital threats like viruses or worms can affect all of the users. Even mobile phones, which are now connected to the internet as well, are starting to provide a target for criminals. It will not be long before there will be a malware program that infects mobile devices as efficiently as current internet worms do with computers. For these reasons, more and more security companies now provide different security solutions for every possible platform.

Recently, there has been an especially large demand for security software protecting virtual machines (VMs). This seems odd at first glance, since not long ago VMs were presented as security solutions themselves. A few years ago, there was more than one paper describing how VMs could improve the digital security within a company and would also allow the company to reduce hardware and management costs at the same time. [GW07,Ros04] However, what people see now is that virtualization is not the ultimate security solution they thought it would be. [VN08] It has to
be understood that virtualization mainly provides isolation, not security in general. Granted, application separation as discussed in chapter two may help to improve the overall security of a system, but it remains just one protection mechanism among many others. In fact, most of the security threats we face in a non-virtualized environment exist in virtualized environments as well. Despite this, virtualization provides some very interesting properties that could be used to create highly secure systems. This is why there is currently a lot of research in this area trying to build security directly into the hypervisor, effectively providing secure virtualization software. These approaches are called secure hypervisors.

This paper will explore and discuss three secure hypervisor approaches: Terra \[GPC+03\], the VAX VMM \[KZB+90\], and sHype \[SVJ+05\]. However, before we start to talk about secure hypervisors, we first have to analyze what security benefits virtualization provides and the threats that virtualization faces. This will be done in section two of this paper. Section three will then provide the reader with an overview about important formal security models that are necessary to understand the secure hypervisor approaches, which will for their part be presented in the fourth section and discussed in the fifth section of this work. Finally, related work will be presented in the sixth section, before a conclusion is drawn in the seventh section of the paper.

2 Background

At this point the reader may still ponder about the question if we actually need additional protection mechanisms for hypervisors. To be able to understand this, we first have to take a look at the security benefits of hypervisors and their shortcomings. This will give us a better idea about the different kind of threats we are dealing with in virtualized environments and the safeguards we need to counteract them.

2.1 Security Benefits of Hypervisors

**Isolation** can, if properly used, be a very important security-relevant property of hypervisors. Isolation guarantees that software running on one of the VMs does not affect the remaining VMs in any kind of way. Whatever enters a VM stays within the VM. Therefore malicious software within a VM can only affect that particular VM, but not the underlying physical hardware, the hypervisor, or other VMs. This also implies that the resource usage of one VM should not affect the performance of other VMs running at the same time. Ideally, each VM should behave the same way it would as if it was running on its own physically separated hardware. Notice that this also means that data that belongs to one of the VMs should by default not be accessible by any other VM even if the information is stored on the same physical hard drive. \[Ros04\]

Isolation as described above is not really a security feature. In fact every hypervisor has to provide isolation in order to be able to use it as a replacement for multiple physically seperated machines. Therefore isolation is actually rather a basic feature of a hypervisor than a security property. However, we can use the isolation that hypervisors provide for application seperation. Hypervisors allow us to place applications in their own VM without the use of additional hardware in a cost, time, and effort efficient way. Therefore we can easily separate crucial or vulnerable applications from the rest of the system by placing them in their own VM. This increases the overall security of the system effectively, since on the one hand the separated applications are obviously harder to compromise using vulnerabilities that may exists in other parts of the system, and on the other hand the system itself remains untouched if one of the separated application is compromised. \[GW07\]
Small Codebase. Since they have to provide a lot less functionality, hypervisors tend to have a much smaller and a lot less complex codebase than modern operating systems (OSs). From this follows that it is much easier to ensure that the security critical code of a hypervisor is error-free as compared to ensuring that a modern OS does not contain any flaws. Therefore hypervisors can be the better choice as a Trusted Computing Base (TCB) compared to a traditional OSs. [Ros04]

Moving Security out of the VM. Finally, hypervisors offer the possibility of moving the security mechanism out of the OS. This would enable protection mechanisms like anti-virus software to guard the system even if a VM is compromised. This is a huge advantage compared to current protection mechanisms that can be circumvented as soon as the OS is compromised. In addition, all virtual machines could be protected from the same external security system. In this case, we would only have to keep this external system up-to-date instead of installing and updating different protection mechanisms in each of the VMs. [RG05]

2.2 Threats in Virtual Environments

Covert channels are a big threat in virtual environments. A covert channel is a way to communicate information to another party using a “hidden” channel that was not designed for communication [JSS07]. For instance, if multiple parties can view the list of files and create new files in a certain folder, a covert channel can be established. The party that wants to transmit data either creates a file in a certain period of time if it wants to signal a bit set to one or does nothing if it wants to signal a bit set to zero. Now the other party only needs to check the number of files in the folder for each time period to receive the information.

In general, covert channels exist whenever two parties have access to a shared variable where one party can write to this variable and the other party can read from the variable. In the example above, this variable was the number of files in the directory. There exist two different kinds of covert channels: storage channels and timing channels. Storage channels modify a stored object, such as the number of files in a certain directory, to transmit information, while timing channels use carefully timed events for the transfer of information. For instance, a process $A$ could try to read a certain page of a public shared library, while a process $B$ could read the same page or other pages of the shared library provoking page faults. Process $A$ can then check how long it takes for him to read the page. If the operation is rather slow it can conclude that the page was not in memory and a page fault occurred. With a page fault being a one and a page in memory being a zero, $A$ and $B$ can transmit information using the above described timing channel.

As one can imagine, detection and elimination of covert channels is a very difficult task. Eliminating all covert channels may even be impossible, since there may be certain variables that all parties need access to like the CPU. Therefore instead of eliminating covert channels it is sometimes more feasible to reduce their bandwidth or to add noise to the channel which may render it unusable. Further it is worth mentioning that Discretionary Access Control (DAC) [Jor87] is no help when it comes to covert channels. To reduce covert channels effectively, Mandatory Access Control (MAC) [LSM+98] is needed that enforces a formal security model. For instance, the covert channel, described in the first example above, could be rendered useless, if users with different access classes could not read from and write to the same directory. From this follows that, only users with the same access classes could establish this particular covert channel, which would be pointless, since they have access to the same information anyway. As we will see in the next section of this
work, formal security models constrain the flow of information between users of
different access classes, thereby effectively avoiding simple storage channels such as
the one described above.

Even it does not seem so at the first glance, covert channels are a very important
topic especially in environments where highly sensitive data is processed and stored.
They also play a crucial role in virtualized environments, since VMs running on the
same physical machine could use covert channels to exchange information with each
other without making use of network connections. This is a big advantage for the
attacker, since network traffic is usually monitored and analyzed, while hardware
resources are often not watched at all. [Gli93]

**Explicit Information Flows.** In most cases, malicious users do not even need to
establish a covert channel to exchange information with each other. Just consider
the possibilities of virtual networks or virtual shared discs, which are features that
most of the hypervisors nowadays provide. Granted, these features make it easier
for legitimate users to do their work, but they can also be easily abused by mali-
cious users to share information that is confidential or to compromise other VMs.
Of course, one could argue that these features shatter the concept of isolation and
should therefore be abandoned. But a system that provides strict isolation is not
really useful in the real world. Therefore it is better to provide these features in-
stead of having the users find their own twisted way of dealing with the problem.
What is, however, definitely needed are better ways to control the explicit flows of
information.

**Single Point of Failure.** Hypervisors are particularly interesting for Denial of
Service (DoS) attacks. Taking out the hypervisor will result in the breakdown of
every VM running on top of it. Even worse: A compromise of the hypervisor will
give an attacker unlimited control over all VMs and their data. Therefore Virtual
Machine Monitors (VMMs) are very lucrative targets for an attacker, especially if he
just wants to inflict as much damage as possible or if important corporate services,
like web servers, are running in VMs. This is one of the reasons why the hypervisor
itself must be as secure as possible [Ren07,Kir07].

**Resource Monitoring.** In a virtualized environment, it may be possible for one
VM to monitor the resource usage of another VM. For example, a VM may be
able to monitor the network traffic of another VM. This is without a doubt a very
serious problem if sensitive data like credit card numbers or passwords are sent in
the clear over the network. It might even be possible to redirect the traffic and to
launch a Man-In-The-Middle (MITM) attack against another VM. Achieving this
on physically separated machines is of course harder, since both machines are not
using the same physical network card to send and receive packets [Ren07,Kir07].
Since a hypervisor provides isolation attacks involving resource monitoring should
be impossible. However, we must not forget that hypervisors are merely software
that may contain bugs like any other software and in fact do as recent studies show
[Orm,Fer06]. Therefore attacks as the ones described above may be easily possible
in comparison to physically separated machines and should be kept in mind.

### 3 Formal Security Models

To constrain explicit information flows and to reduce the possibilities for covert
channels, we need formal security models with solid mathematical foundations.
These security models can then be enforced by the hypervisor using MAC. This will
increase the overall security of the system significantly, since it allows the hypervisor to protect the confidentiality and integrity of information, even if it is shared among multiple VMs. Therefore this section will discuss three well-known formal security models, which are also used by some of the secure hypervisor approaches that will be presented in the next part of the paper.

### 3.1 Bell-La Padula

Among the various security models that have been published over the years, the Bell-La Padula model [BP76] is probably the best known. The model is named after its creators and was originally an attempt to develop a “mathematical model of security in computer systems” [Bel05] that could be used to protect government information, which is categorized in different security classifications such as “secret” or “top secret”. However, the Bell-La Padula model is only concerned with confidentiality, but not with the integrity of the data.

The security model considers two basic entities: subjects and objects. Subjects are defined as active entities such as users or programs acting on behalf of a user, whereas objects are defined as passive entities like data or files. Each subject and each object have a certain “security designation”. This security designation is a pair consisting of a classification and a set of categories. Thereby the classification states a security clearance such as “unclassified”, “classified”, “secret”, or “top secret” and the set of categories defines a subset of formal categories that are associated with the information. Given these security designations, two rules are used to decide whether a subject is allowed to access an object or not. The first rule is called the “simple security property” (ss) or “No-Read-Up” rule. It says that a subject can only read an object if its classification is higher or equal to the classification of the object and the category set of the object is a subset of the category set of the subject. In this case the subject “dominates” the object.

However, the ss rule is not enough to guarantee the confidentiality of the data, because a subject that can read “top secret” data may write this data to an object with a lower classification - for example “secret” - thus effectively breaking the security model. Therefore a second rule, the so-called “*-property” or “No-Write-Down” rule, was specified. It says that a subject can only write to an object that dominates the subject. From this follows that a subject cannot declassify objects, which solves the above-mentioned problem. This rule is also the reason why the Bell-La Padula model does not guarantee the integrity of data, since a subject with a lower classification can write to an object with higher classification.

As already mentioned above, the Bell-La Padula model was intended for military and governmental use as is heavily reflected in its design. All rules are based on the “Need-to-Know” principle, which says that a person should only have access to information that he needs to do his job. Therefore the primary objective of the model was to constrain the flow of information from the lower clearance levels to the higher clearance levels. [BP76,Bel05]

### 3.2 Biba

A year after the Bell-La Padula model [BP76] was proposed, Kenneth J. Biba suggested another closely related security model called the Biba model [Bib77]. In contrast to the Bell-La Padula model, which is concerned with the confidentiality of data, the Biba model tries to guarantee the integrity of data.

The Biba model considers also subjects and objects, which are defined in the same way as they are in the Bell-La Padula model. Furthermore it also uses two rules to decide whether a subject is allowed to access an object or not. The first rule is called the “simple integrity property” or “No-Read-Down” rule. In contrast to the simple
security property of the Bell-La Padula model, the simple integrity property says that a subject can only read an object if it is dominated by the object. For instance a person is only allowed to read information which has the same clearance level as the person or a higher clearance level than the person. Further the “*-property” of the Biba model, which is the second rule, is also defined as the opposite of the *-property in the Bell-La Padula model. It says that a subject is only allowed to write to an object that is dominated by the subject ("No-Write-Up"). Therefore a person can only write to a file that has the same clearance level as the person or a lower clearance level than the person.

As one can see, the Biba model is actually just an inversion of the Bell-La Padula model. It follows the principle that information with a high clearance level has a high level of integrity and therefore this information can only be written to lower clearance levels. In addition persons should not be allowed to read information on a lower clearance level than their own, since people with a lower clearance level are allowed to write this information and therefore the level of integrity of the information decreases. [Bib77]

3.3 Chinese Wall

The Chinese Wall security model [BN89] was first proposed in 1989 by Brewer and Nash and is therefore sometimes also called the Brewer-Nash-Model. In contrast to the Bell-La Padula model [BP76], which was primarily developed for the U.S. government and the U.S military, the Chinese Wall model is intended for use in the financial sector and tries to protect the integrity and confidentiality of data at same time. It deals with the problem that an analyst, who possesses insider knowledge about his clients, should not be allowed to advise a competitor of one of his clients. However, the analyst should, of course, be able to advise any other company that is not a competitor of one of his clients. From a technical point of view this means that a user should not be allowed to access information which is in conflict with information he already has access to. This is called the “simple security rule” of the Chinese Wall model. To enforce this rule we consider individual items of information, which are also called “objects” in the Chinese Wall model. Objects are grouped into “company datasets” according to the company they belong to. Finally, we define “conflict of interest classes”, which is a list containing all companies that are in conflict with each other. If a user now tries to access a dataset of a company, it has first to be verified that he has, in fact, never had any access to any dataset of a company that is in the same conflict of interest class as the company he is now trying to access.

It turns out that the simple security rule of the Chinese Wall model is again not enough to guarantee the confidentiality and the integrity of company information. Consider, for example, two users $U_1$ and $U_2$ and three companies $C_1$, $C_2$, and $C_3$. $C_1$ is in conflict with $C_3$, but $C_1$ is not in conflict with $C_2$ and $C_2$ is not in conflict with $C_3$. $U_1$ has access to $C_1$ and $C_2$, whereas $U_2$ has access to $C_2$ and $C_3$. In this setting it would be possible for $U_1$ to read data from $C_1$ and write it to $C_2$. This information would then be accessible by $U_2$ and he could therefore access information that is in conflict with information he already possessed. To thwart this attack or to prevent the accidental disclosure of information, the write access of users has to be constrained as well. This is done by the *-property of the Chinese Wall model. The *-property says that a user can only write data if access to the write destination is allowed to the user by the simple security rule. Furthermore “no Object can be read that is in a different company dataset to the one for which write access is requested and contains unsanitized information”. [BN89] Here “unsanitized” information is information that can be traced back to a certain company. This means that users are allowed to write Objects that represent
information that cannot be associated with a certain company. To determine if information is sanitized or not is strongly related to the database inference problem, which is beyond the scope of this paper. The interested reader is referred to [Far00] for more information.

4 Secure Hypervisor Approaches

In this section three different secure hypervisors approaches will be presented. Each of the approaches will be described in turn. Thereby the paper will mainly focus on the concept behind each of the approaches, but will not discuss their particular implementation in detail. A comparison of the approaches including a discussion of the advantages and disadvantages of each approach can be found in the following section of the paper.

4.1 Terra

In 2003 Garfinkel et al. published a paper presenting a virtual machine-based platform for Trusted Computing called Terra [GPC+03]. Particularly, Terra provides a “flexible architecture […], that allows applications with a wide range of security requirements to run simultaneously on commodity hardware.” [GPC+03]. This is achieved through the use of a Trusted Virtual Machine Monitor (TVMM), which offers, in contrast to traditional hypervisors, advanced protection mechanisms. These mechanisms, which will be described in detail shortly, allow Terra not only to support general-purpose computing platforms, but also high-assurance systems. Since an increase in system security does usually not take place without decreasing the performance of the system and also because a general-purpose system has different security requirements than a high-assurance system, we should be able to control whether these additional protection mechanisms have to be enabled for our VM or if we can do without them. The authors approach this problem by offering two basic VM abstractions that have to be supported by the TVMM: “open-box VMs” and “closed-box VMs”. In contrast to closed-box VMs, the privacy and integrity of open-box VMs is not specially protected by the TVMM, leaving open-box VMs virtually indistinguishable from VMs running on other hypervisors like Xen [BDF+03]. Closed-box VMs on the other hand are guarded by the TVMM using three additional protection mechanisms that are not available in traditional VMMs:

Root Secure: Only the creator of the closed-box VM can access and change it. From this it follows that the privacy and integrity of a closed-box VM is even protected from the administrator of the TVMM.

Attestation: Through attestation a closed-box VM is able to identify itself in a reliable way to a remote party. To accomplish this a so-called “chain of trust” is established starting from a tamper-evident hardware platform. Among this tamper-evident hardware is a tamper-evident chip that contains the private key of the hardware signed by the vendor of the machine. On system boot the tamper-evident hardware certifies the firmware, which certifies the boot loader, which in turn certifies the TVMM, which continues the certification process with the VMs. In particular this certification process consists of the following steps for each component:

1. The component that wants to be certified generates its own public/private key pair.
2. Afterwards the component hands his public key and any additional application data it wants to have signed to the lower-level component using the “ENDORSE” API call.
3. The lower-level component uses his private key to sign a certificate that contains the public key and any additional application data that it received as well as a hash “of the attestable parts of the higher-level component” \cite{GPC03}. For example, if a TVMM signs a VM this hash “includes the BIOS, executable code, and constant data of the VM” \cite{GPC03}.

![Fig. 1: Terra: Remote Attestation](image)

At the end of this certification process each VM has obtained a certificate chain describing the whole software stack consisting of the tamper-evident hardware, the firmware, the boot loader, the TVMM, and the VM. If a remote party now receives this certificate chain it can verify the given information and afterwards decide if the other system is trustworthy depending on the data it received. For example, a remote party could have a list of all trusted VM hashes and could therefore deny the communication to a VM if its hash is not contained in the list.

**Trusted Path:** The TVMM ensures that there is a trusted path from the user to the applications, meaning that a user can identify the VM he is interacting with and that the VM can identify that it is communicating with a human user. On top of that the TVMM also protects the integrity and privacy of communications over this trusted path.

It is obvious that the above mentioned protection mechanisms increase the security of closed-box VMs by far compared to traditional open-box VMs, since closed-box VMs can, for instance, decide if a remote party is trustworthy using remote attestation. Unfortunately, a tamper-evident hardware chip is not yet shipped with every computer. This effectively limits the number of machines that Terra can be run on, if attestation, which is one of the central security aspects of the system, should be used. However, this may change in the near future. According to IDC more than 250 million notebooks and desktop computers will be shipped with a TPM by 2010 \cite{vB}.
In addition to the protection mechanisms which are provided by the TVMM, Terra also requires a Management VM. The purpose of this Management VM is to manage all platform resource policies and access controls. To fulfill its tasks, the Management VM can use an interface exported by the TVMM that provides many important operations like starting or stopping a VM. In general, however, the Management VM is independent of the TVMM, which implies that a platform administrator could use any kind of OS as Management VM as long as it performs the given duties.

In conclusion one can say that Terra is a very flexible architecture that provides some very important security features. One of them is attestation that allows a VM to establish a secure communication channel with another party. However, to be able to use all of the security features that Terra offers, the system has to be run on tamper-evident hardware.

4.2 VAX VMM

One of the first attempts to design a secure hypervisor was made by Karger et al. \[KZB^{+}90\] in a research project dating back to the period between 1981 and 1990. This research project focused on the development of a production-quality VMM security kernel that was “capable of receiving an A1 rating from the National Computer Society Center” (NCSC) \[KZB^{+}90\]. The A1 rating was the highest security rating in the Trusted Computer System Evaluation Criteria (TCSEC) \[Bra85\] published by the NCSC in 1985 and is also known as the “Orange Book”. Essentially, a system that achieves an A1 rating according to the TCSEC has to have DAC as well as MAC and must be formally verifiable.

The development of this VMM security kernel was carried out on the virtual address extension (VAX) architecture designed by Digital Equipment Corporation during the 1970s. This is why the VMM security kernel of Karger et al. is often also called the VAX security kernel. From this point on the terms VAX hypervisor and VAX VMM will be used interchangeably to refer to the VAX security kernel.

As demanded by the A1 requirements, the VAX hypervisor supports DAC as well as MAC for all VMs. In particular, the VAX VMM MAC enforces the Bell-La Padula \[BP76\] model for privacy protection and the Biba \[Bib77\] model for integrity protection, which have been described in section three of this work. To be able to support these security models, the VAX hypervisor distinguishes between subjects...
and objects. A subject is either a user or a VM, while an object is defined as basically everything but a user such as a real device, volumes, or security kernel files. Each subject and each object is assigned a secrecy class as well as an integrity class, consisting of a secrecy level and a secrecy category set or respectively of an integrity level and an integrity category set. These classes are used by the system to decide whether a subject is allowed to access an object or if access has to be denied. This decision is based on the rules of the security models, which have been explained in section three.

Whenever a user wants to access a VM he has first to authenticate herself to the VAX VMM. For this purpose the VAX hypervisor offers a trusted process running in the kernel that is called the “Server”. This process only executes verified machine code and does not accept any user-written code. If a user wants to interact with the VAX hypervisor a trusted path between a server process and the user is established. The server provides commands that allow the user to connect to a VM depending on his access rights. In case the user has the necessary rights to connect to a VM another trusted path is established between the user and the VM, allowing him to interact with the OS running in the VM.

Furthermore, the VAX hypervisor was carefully analyzed for covert channels throughout the whole design and implementation process. For this purpose various techniques were used by the authors including informal reviews by the system engineers, code-level flow analysis, and “fuzzy time” to reduce the bandwidth of timing channels [KZB+91]. Interestingly, most of the covert channels in the system were discovered through careful informal reviews by the system engineers. According to the authors “timing channels proved to be a much more serious problem” [KZB+91] than storage channels, since “many of them were inherent in the underlying hardware” [KZB+91]. Above of that, it turned out, that the bandwidth of storage channels was usually lower than the bandwidth of storage channels. To solve the problem the authors used various, not further explained, complex concepts, one of them being “fuzzy time”, which involved fuzzing all existing clocks in the system, to reduce the bandwidth of timing channels.

In summary the VAX VMM was a very interesting project that tried to meet a very high security standard. This security standard demanded not only the enforcement of formal security models, DAC, MAC, and the analysis of covert channels, but also required auditing, a verifiable design, secure distribution to the end user, and a high level of assurance. Unfortunately, covering each of these requirements in turn or describing the architecture of the VAX hypervisor and its development process in detail would be far beyond the scope of this paper. However, the project shows that there is much more to secure hypervisors than a good system design. It also has to be verified that the system itself is implemented correctly and does not contain any serious flaws that can compromise the security of the whole system. However, this is not an easy task as everybody who ever developed a program knows.

4.3 sHype

The sHype security architecture is without doubt one of the best-known approaches when it comes to creating a secure hypervisor. sHype evolved from an IBM research project and was originally developed for IBMs rHype, an open-source research hypervisor, but was shortly after its first release also implemented for the well-known Xen [BDF+03] open-source hypervisor. The primary goal of the project was to develop a way to control the information flows between VMs. [SJV+05,LeM06] However, before we start to discuss the sHype concept in more detail, it is important to know that sHype does not attempt to control all information flows among VMs, but only all explicit information flows. This implies that sHype does not try to eliminate covert channels despite the fact that its architecture minimizes the possibilities for
covert channels significantly.

To control the information flows between VMs, also called “logical partitions” by the sHype authors, sHype uses MAC to enforce a formal security policy. Particularly, sHype uses the concept of a “reference monitor”. In a nutshell, a reference monitor “enforces the authorized access relationships between subjects and objects of a system” [SVJ+05]. This means that the reference monitor is called whenever a subject wants to access an object and is responsible for granting or denying the access attempt based on the security policy. However, the reference monitor usually does not decide whether a subject can access an object. It only enforces the decision, which is often made elsewhere in the system.

In the sHype architecture, the so-called “Access Control Module” (ACM) is responsible for this decision. The ACM uses the formal security policy, the labels, which are attached to the subjects and objects of the system, and the type of operation a subject wants to execute to make an “Access Control Decision” (ACD). Therefore the complete workflow that the system executes if a subject tries to access an object is as follows: The access call of the subject is intercepted by the reference monitor, which in turn calls the ACM by placing an “Authorization Query” (AQ). This AQ contains the labels of the subject and the object and the operation that the subject wants to execute (read, write,...). The ACM uses the formal security policy and the data of the AQ to make an ACD, which is then returned to the reference monitor. Finally, the reference monitor enforces the ACD by allowing or denying the Subject to execute the operation.

In this process, the reference monitor is actually implemented using so-called “Enforcement Hooks”, which are distributed throughout the hypervisor. Whenever a subject tries to access an object, an enforcement hook is triggered and the ACM is called.

As one can see, the architecture of sHype is very flexible, since the ACM is independent of the reference monitor. In general, it is good practice to separate policy enforcement and policy management. If changes are made to the ACM, the reference monitor can stay the same as long as the interfaces between the ACM and the reference monitor stay untouched, which is usually the case. Of course, the same also applies to the ACM if the reference monitor is changed.

sHype defines a subject as a Partition (VM), while an object is a virtual resource like a virtual disc. From this follows that sHype only controls the access of VMs to

Fig. 3: sHype Architecture.
virtual resources. sHype neither controls nor cares which program within the VM is trying to access the object. This problem is left to the VMs. If it is stipulated that only certain programs or processes are allowed to access an object, then the VM itself has to enforce these rules.

To specify and manage the formal security policies used by the ACM, the authors of sHype propose a privileged VM called the “Security Policy Manager Partition”. The purpose of this VM is to make the handling of security policies easier for the system administrator and to avoid errors in the policy files. This could be a program, for example, that provides the administrator with a predefined mask allowing him to create formal security policies faster and easier. Further the policies could also be automatically verified before they are applied by the ACM.

Since sHype is capable of interpreting and using formal security policies, it can basically support any security model. The sHype architecture already supports many formal security models including Bell-La Padula [BP76], Biba [Bib77], and Chinese Wall [BN89], which have been presented in section three. This not only makes the architecture very flexible, but also has the advantage that the security policies are verifiable and independent of the technical implementation. In addition, this approach makes it a lot easier to detect and protect against covert channels, since most of the formal security models have been examined for the possibilities of covert channels and how they can be avoided.

Taking everything into consideration, one can say that the sHype architecture is very flexible and provides support for a wide range of different security models. sHype uses formal security policies created according to these models to control the explicit flows of information between VMs. However, the architecture still has problems with covert channels, the rectification of which was beyond the given scope of the sHype project.

5 Evaluation

This part of the paper will compare and discuss the three different secure hypervisors approaches described in the previous section. For this purpose four categories have been selected that cover important security aspects as well as generally important architectural properties. These categories are: Objective (Did the approach achieve the goal of the project?), Overall Security, Flexibility, and Practicability. Each of these categories will be discussed in turn, before a summary is presented at the end of the section.

5.1 Objective

One of the most important questions we have to answer is, of course, if each of the approaches achieved the goal it was aiming at. This question will be answered below.

**Terra.** The main goal of Terra was to provide a flexible architecture that would support applications with a wide range of security requirements and would despite that still run on commodity hardware. Terra tries to achieve this goal using a closed-box/open-box approach that can support very secure VMs running in a closed-box environment as well as conventional VMs running in an open-box environment. Beyond that, Terra also provides remote attestation, which can be used by remote applications to obtain information about the host they are interacting with. The so obtained information can then be used to decide whether the host machine is trustworthy or not, which increases the security of the remote application further.

In conclusion, Terra provides indeed a very flexible concept that can be used as a
platform for a wide variety of systems with a wide range of security needs. On top of that, the only requirement for Terra is a tamper-evident hardware chip, which is already part of most modern desktop computers and notebooks and will be shipped with 90% of the systems by 2010 [vB]. Unfortunately, however, there is no complete implementation of Terra yet. From this follows that, Terra achieves its primary objectives, but just in theory.

**VAX VMM.** In contrast to Terra, the focus of the VAX hypervisor project was primarily on achieving a high security standard, since the main objective of the VAX VMM approach was to meet the A1-level security requirements of the NCSC. This goal was met after almost ten years of development time and multiple implementations of the system when the project was finally canceled in 1990. Nevertheless, the authors claim that the project was a technical success and that there was a huge customer demand for it. But because of export restrictions for B3-level and A1-level systems and other confidential reasons, the project had to be discontinued in the final analysis anyway. [KZB+91]

**sHype** was mainly developed with the intention of controlling all explicit information flows among VMs. In addition, the authors also define content integrity guarantees, content attestation, and secure services as other long-term security goals of the project. [SVJ+05] So far sHype has proved capable of managing explicit information flows and providing content integrity guarantees using MAC enforcing formal security policies. Further there has been recent promising work when it comes to controlling covert channels using sHype [JSS07] and enabling remote attestation for sHype [MBC+06]. Therefore sHype is not able to meet all of its security goals at this point, but will probably do so in the near future.

### 5.2 Overall Security

Not all of the projects discussed in this paper defined VM security as their primary objective. However, when it comes to the subject of secure hypervisors, VM security must obviously play a decisive role. Therefore we should take a closer look at the overall level of security that each of the approaches provides.

The **VAX hypervisor** approach provides without doubt the highest degree of security among the different approaches presented in this paper because of the A1-level requirements of the NCSC. These requirements do not only demand DAC and MAC, but also require formal verifiability, analysis of covert channels, enforcement of formal security models, an integrated Intrusion Detection System, auditing, and much more. On top of these security features there are also multiple obligations for the software development process including reviews and secure coding standards. Therefore the VAX VMM that primarily tried to meet these requirements has of course a very high security standard that cannot be matched by Terra or sHype. However, this does not imply that Terra or sHype are insecure. On the contrary both approaches provide some very essential security properties. Terra, for example, provides attestation, a feature that even the VAX hypervisor is not capable of, but which is becoming more and more important nowadays, in times when almost every machine is connected to some sort of network. Therefore it is, of course, not a coincidence that the sHype project has also work in progress aiming at attestation.

Unfortunately, in contrast to sHype and the VAX VMM, Terra does not provide MAC, despite the fact that this constitutes a very important security feature. Instead Terra leaves the enforcement of security policies and access controls to the
Management VM without specifying which policies have to be supported and in which way they should be enforced. Therefore Terra is not able to constrain explicit information flows reliably. This enables Terra to remain a very flexible architecture as required, but weakens the overall security of the architecture significantly. However, we must not forget that MAC in combination with a formal security policy could also be provided by the Management VM or the VM itself, which may allow adapting the security policy even better to the needs of the specific VM. Yet, just the fact that Terra is not necessarily capable to enforce a formal security policy using MAC itself, sparks security concerns. Furthermore, we cannot say anything about the codebase of Terra, since there is no complete implementation of Terra yet.

Lastly, sHype provides a very flexible MAC that is capable of enforcing many different formal security models. This allows sHype to constrain all explicit information flows and also reduces the number of covert channels in the architecture. Unfortunately, covert channels still remain an issue in sHype, a problem which, however, may be resolved in the near future when one takes into account recent work of the sHype authors. [JSS07] In addition, sHype has a very small codebase and contains therefore probably only a small number of flaws if any. In conclusion sHype achieves a decent security level that will be improved significantly by the work that is currently in progress.

5.3 Flexibility

In addition to security, flexibility is another important factor of a system. In general, there is no use in having a very secure system if it cannot be used by many people, because it is restricted to a certain environment or to complex. Of course, this may not apply to every scenario, but at least two of the secure hypervisor approaches discussed in this paper were developed with flexibility being a primary or secondary objective.

Unfortunately, there is usually a trade-off between security and flexibility as can be seen by the VAX VMM approach. The VAX VMM provides a very high level of security, but it is restricted to the VAX platform and does not even support all of the VAX processors. [KZB+90] Therefore although the VAX hypervisors achieves his primary goal of providing a high security standard, it remains inflexible and may at the end only be suited for highly secure environments. But having said that, we have to keep in mind, that the VAX VMM was actually never intended to be very flexible in the first place.

Terra on the other hand has a few security concerns, but provides a very flexible concept that could be adapted to almost any system. It relies, however, on certain hardware requirements, like tamper-evident hardware, to be able to make use of its complete functionality. This is not a serious problem, since tamper-evident hardware becomes more and more widespread nowadays.

Lastly, sHype seems to lie somewhere between the VAX VMM and Terra, trying to find a good trade-off between security and flexibility. sHype provides a flexible architecture that allows it to be implemented for a wide range of different hypervisors. Currently, it is already supported by the rHype research hypervisor [SVJ+05] and by the popular Xen hypervisor [BDF+03], which itself supports a wide variety of OSs and hardware. However, sHype has to be reimplemented if you want to use it in conjunction with a different hypervisor.

5.4 Practicability

Concepts and theoretical models are a very good thing. They allow us to verify our approach before we start to implement it. However, we have to keep in mind that
not all theoretical models can be implemented that easily. In the worst case it may even be impossible. Therefore it is important that the concepts presented in this work are not only flexible and secure, but can also be implemented in a reasonable amount of time using a reasonable amount of resources. Of course, the reasonable amount of resources always also depends on the intended purpose of the system. Consider the VAX VMM approach, for instance. It took almost ten years to design and implement it. This is at least a questionable amount of time for a secure hypervisor that may be very secure, but can only be used in conjunction with a certain architecture. Although it might be possible to speed up the process, if the lessons learned from the project are taken into account, it seems unlikely that the high level of security is worth the trouble for anything, but highly secure environments, which may also be one of the reasons, why the project was finally canceled in 1990. However, in the end we must not forget that the project, from a practicability standpoint, showed that it is indeed possible to develop such a secure system given the right amount of time. This is not a matter of course considering the huge number of requirements for an A1-level secure system stated by TCSEC.

In contrast to sHype and the VAX VMM, a complete implementation of the Terra architecture does not exist, which is a big problem. The authors of Terra implemented their prototype using VMware GSX Server 2.0.1 and Debian GNU/Linux, even though they stated that “neither Debian nor VMware GSX Server is suitably high assurance for a real TVMM” [GPC+03]. This raises a lot of questions about the practicability of Terra, since so far there has been no decent implementation of the architecture that would support the claims of the authors or give indications about the performance of the system.

Finally, there remains sHype which has produced good practical results to date. sHype’s already existing implementations for multiple hypervisors have shown that the system can be ported to a different platform in a relatively small amount of time. For example, the authors claim that the sHype implementation of Xen [BDF+03] has only 2600 lines of code. [SJV+05]

5.5 Summary

Terra is, all in all, a very promising approach, but a lot of work still has to be done before the architecture can be used in a commercial environment. At the moment, the main problems that Terra has are the missing implementation and the lack of mandatory enforcement of formal security policies. It is crucial that these problems be resolved as soon as possible to be able to provide an architecture that can persist in the real world. However, once these issues are resolved, Terra could be a huge success, since it can be adapted to almost any security environment without changing its source code.

The VAX VMM may be an old project, but it is nonetheless very important. This project clearly shows how hard it is to create a highly secure system that can be run on commodity hardware. Besides the design of the system, the whole software development process has to be aligned to high security standards, which increases the development time significantly. In addition, the complexity of the system usually increases with higher security standards, which makes the system harder to use in everyday life and may be also a reason why the project was canceled. Nonetheless, the VAX hypervisor is a very secure system that provides a good platform for high-assurance applications. Further the lessons learned from this approach could be very useful when it comes to designing a secure system or while trying to find the right measure between security and usability.
sHype provides a good ratio of flexibility and security and is, in addition, reasonably small and well thought-out to be practicable. However, sHype is so far not eligible to support high-assurance applications due to its lack of control over covert channels and its inability to provide remote attestation. The work currently in progress might rectify these problems and will hopefully allow sHype to provide a higher security standard while still maintaining its flexibility.

6 Related Work

The author is not aware of any work that is similar to this work and that compares several secure hypervisor approaches with each other. However, there exists a lot of different work comparing the security of hypervisors in general. Ormandy [Orm] and Ferrie [Fer06], for example, analyzed implementations of popular VMs using source code auditing, known attacks, and fuzz testing to validate the security of the implementations. Surprisingly, they discovered that all of the tested hypervisors contained a flaw of some kind. This proves that there is a severe need for secure hypervisors.

King et al. [KCW+06] explored the possibility of using hypervisors as rootkits. Once the target OS is compromised, a hypervisor is installed and the original OS is run in a VM from there on. This makes it virtually impossible for security software running on top of the OS to detect the rootkit. Since a secure hypervisor would run below the OS, the compromise of the OS could be detected and avoided, effectively scotching the attack.

In addition to that Govindavajhala and Appel [GA03] present a very interesting attack on the Java Virtual Machine (JVM) and .NET VM using memory errors allowing them to compromise the system. From this it follows that even VMs like the JVM, which were designed with security in mind, are vulnerable to certain attacks. This confirms how hard it really is to design a secure system.

Payne et al. [PSC+07] have done research into an interesting way of reducing the complexity of MAC throughout the system using a hypervisor approach. This could make the management of formal security policies a lot easier and would also provide finer grained access controls on each access level.

Finally, there are other interesting approaches to secure hypervisors like NetTop [MS00], Bluelane’s Virtual Shield [Tec], VMware’s VMSafe [VMW], and Reflex’s VSA [Ref] that could, for reasons of time constraint and lack of availability of neutral information, not be adequately discussed in the confines of this work.

7 Conclusion

This paper presented three different approaches to secure hypervisors: Terra [GPC+03], the VAX VMM [KZB+90], and sHype [SVJ+05]. All of these approaches contain important concepts and ideas. However, none of them is as yet ready to be used in everyday life for various reasons. The VAX VMM is restricted to the VAX architecture, which is not in use anymore, while Terra has not been implemented yet and involves some security issues that have to be resolved before it can be used. sHype on the other hand has already been implemented for Xen [BDF+03], a very popular open-source hypervisor, but some crucial security features are still in development. An interesting way to solve many of these problems would be to combine Terra and sHype into a single architecture. This would lead to a very flexible, powerful, and above all secure hypervisor that would be suited for use in almost any environment.

The current state of affairs is that a lot of work still has to be done before a secure hypervisor solution is available for public use. Till then administrators of virtual
environments will just have to attempt to secure each VM individually using existing security solutions for non-virtualized systems or using an architecture similar to that described in the NetTop [MS00] approach. Let us hope, however, that it will not take too long, since secure hypervisors could be a major step towards more secure systems.
References


