Using Trusted Computing to Detect VM Rootkits

Ankur Sahai

University of Southern California

asahai@usc.edu

Abstract

Virtual Machine (VM) Rootkits have emerged as an important security threat recently as they are difficult to detect. A VM Rootkit works by installing itself at a lower abstract logical layer in the machine, closer to the kernel; thus making it difficult for the security tools running at a higher logical layer to detect it. The latest and more effective Rootkits, after having intruded the system by exploiting a security flaw, drop a Virtual Machine Monitor (VMM) underneath the Operating System (OS) installation and hoist the native OS into a VM running on top of the malicious VMM as a guest OS. This gives the Rootkit more control over the system and at the same time makes it extremely difficult to detect it; especially by the security tools running on the guest OS (originally native OS) on top of the malware VMM as, it can easily misguide such tools by intercepting the hardware calls made through the guest OS. We suggest a Trusted Computing based technique, that uses remote attestation by having a trusted third party (that has a code of the VM Rootkit) authenticate the code running on the hardware. We analyze the different security issues posed by the suggested technique and explore the security mechanisms that can be used address these issues. We think that, this technique can prove to be very effective in detecting generally any malware present on the system; specifically VM Rootkits that are very difficult to detect otherwise.

1. Introduction

VM Rootkits have received much attention of late by researchers working in the area of malicious software (malware) design and detection. This can be attributed to the interesting behavioral features of the VM Rootkits, primarily their method of operation that makes it very difficult to detect their presence. The term Rootkit was first associated with the set of recompiled Unix commands that would hide the presence of an intruder and allow him to maintain root access to the system. However, the current usage of the term Rootkit refers to the malware that try to hide themselves from the system

Rootkits can be classified into 4 different categories based on the abstract layer in the system where they operate from namely: application, library, kernel and virtual level. Application level Rootkits usually modify or replace the code of the applications running on the system. Library level Rootkits modify the system calls to hide their presence. Kernel level Rootkits modify the kernel code (like loadable kernel modules and device drivers) to hide a backdoor installed on the system. Virtualized Rootkits are the lowest level of Rootkits that work by either modifying the boot sequence of the machine to load themselves instead of the native OS or by installing themselves underneath the native OS.

Rootkits are classified into different categories depending upon its interaction with the system components by Jonna [1]. Type I Rootkits work by modifying the static components of the system like in-memory code sections of the kernel. Type II Rootkits work by modifying the dynamic resources of the system like data-sections e.g. modifying the function pointer in some kernel data-structure, so that, the intruders code now runs instead of the original system or application code. Type III Rootkits usually drop a VMM under the native OS and hoist the native OS onto a VM running on top it.

Application, Library and Type I Rootkits can be detected based on some prior information about their behavior. However, it is very difficult to detect the other types of Rootkits namely Kernel, Virtual level and the Type III Rootkits as they are present at a level below the native Operating System, above which, most of the security tools operate.

We propose a Trusted Computing based Remote Attestation approach where-in the client sends a copy of code running on it to a trusted authentication server. The trusted authentication server looks for VM Rootkits in this code and sends back the result to the client. There are several issues to be addressed here like How does one compare such a large code as the one running on the hardware with the code of the VM Rootkit? We try to address these and related issues in sections 4, 5 and 6. Before that, we give a brief introduction of VMMs and VM Rootkits in Section 2, 3 and Trusted Computing in section 4.

2. VM and VM Rootkits

A Virtual machine is an abstract machine that consists of a part or all of the resources of its host machine. Thus it is possible to have multiple Virtual Machines running on a machine that share the entire resource pool of the host between them. To manage the different virtual machines running on the same physical machine an abstract layer called VMM (Virtual Machine Monitor) also known as hypervisor is added. The Virtual Machines instead of communicating with the hardware directly, now communicate with the hardware through a VMM or hypervisor. All the system calls or traps are made to the Hypervisor which then handles these system calls appropriately.

As is seen from Fig. 1, the VM Rootkits exploit this closeness of VMM to the hardware. They drop in a VMM and have the native OS run on a VM on top of the VMM (as the Guest OS). Now, the VMM is in total control because any system calls made by the applications running on the Guest OS can be trapped before sending them to the underlying hardware layer. So any security tool like a malware detection tool has no way of knowing that it is now running on top of a VMM instead of the original machine.

This means that the VMM has control over most of the resources that can be accessed by the Guest OS. This is a much more dangerous issue than any sort of malicious attack on the OS itself. The Rootkit can run any malicious code on the machine as long as it is not detected by the Guest OS running on a VM. It can either use another OS to run the malicious ode or have some malicious code that does not need to run in an OS.

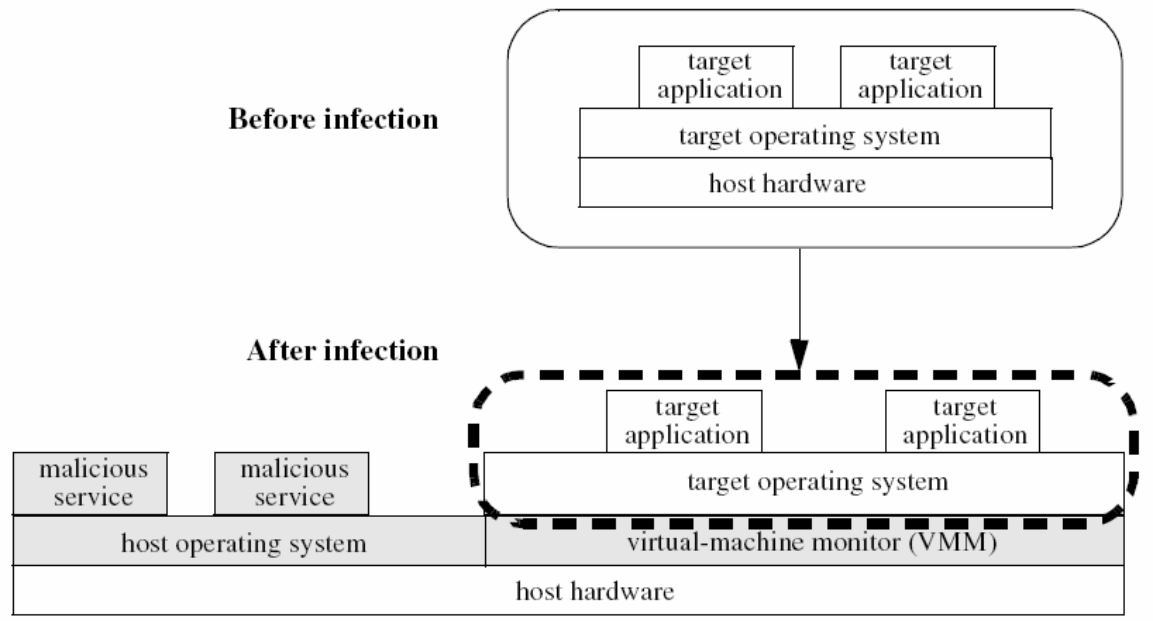

Fig 1. Logical layout of the system before and after the VM Rootkit attack (Courtesy [1])

3. Techniques for detecting VM Rootkits

It is to be noted that running VMM on a system has an associated CPU and Memory and Storage overhead. Out of these CPU overhead is the most reliable source for detection. It is very difficult to detect anomalies in the memory and storage resources of the machine due to their large sizes and more largely random behavior. For example, the VM Rootkit adds an overhead for trapping the instructions from the VM and then communicating with the hardware as compared to the OS running on the VM directly communicating with the hardware. The VMM will also have a slight memory and storage overhead for example the paging overhead due to the VMM pages that have to be in the memory for most of the time which reduces the number of page frames for pages of other processes. Also since the VMMs use virtual devices, significant change may be observed in the devices.

Detection tools that monitor the time for computation of Operating System instructions can be used. Such a timing difference can be captured by comparing the running-time of the benchmarks against an external clock time (as the local clock of the machine can no longer be trusted).

Special instructions may be added to the instruction set that do not trap to the VMM; instead communicate directly with the underlying hardware. For example, instructions that can request logging of the system calls, so that, this log can be subsequently retrieved to look for anomalies indicating the presence of a Rootkit.

All the above mentioned techniques have associated with them, the problem of false positives and negatives because, it may be difficult to definitively say if the overhead is due to a VMM or one or more resource intensive processes.

Abstract security layers may be added closer to the hardware so that the VM Rootkits are not able to run above these. However, the down side with using such tools is that they will require separate computation, memory and storage modules embedded in the hardware which is a very costly approach. They cannot use any of the components above the hardware layer as there is the risk of the VM Rootkits getting in control of them.

Finally a remote attestation technique can be built on top of Trusted Platform Module that can be used to detect the authenticity of the code running on the machine. We propose a protocol based on this technique.

4. Trusted Computing

Trusted Computing is computer security concept that advocates building systems in which at least a minimum level of trust that can be placed. A certain component of the system known as ‘root of trust’ can be always trusted. This ‘root of trust’ acts a starting point for building a ‘chain of trust’ among the different components of the system that can be extended even outside the system. It is based on two fundamental concepts: Endorsement Key and a Root of Trust.

The Endorsement key is public-private key pair where the private key is the secret key embedded in a system component like CPU or Motherboard, so that; it cannot be accessed by other components or processes. Root of Trust is usually implemented as a computing engine embedded in the system that must perform a minimal set of functions independent of the software running on top of the system. Root-of-Trust component should be unique for a system so that the system can be uniquely identified.

Using the PKI (Endorsement Key), it should be possible to identify the system uniquely. This may be helpful for the Remote Attestation mechanism where the system has to interact with a trusted third party to authenticate a piece of code running on it. It is assumed that the trusted third party has the codes of malwares against which it can compare the code send by a machine suspecting malware.

5. Using Trusted Computing to detect VM Rootkits

Security tools running on the suspected system itself cannot be trusted completely especially if they are running on the malicious VMM installed by the VM Rootkit. This is because, the VM Rootkit might have modified the code used by the security tool to hide itself. Thus it becomes necessary to use a trusted authentication server to validate the software running on the system. However, to have the trusted authentication server authenticate the validity of the code running on the machine there has to be built-in

mechanism in the machine to communicate with the authentication server. Moreover, the ability of the machine to communicate with a n authentication server should not be affected by any process running on the machine. This is where a trusted computing based approach fits very nicely.

A root-of-trust module has to be embedded into the Trusted Platform Module. The TPM needs to have a capability to get a snapshot of all the code running on the system and use a hash function to get the hash of the code for integrity. It should then be able to encrypt this using its private (Endorsement) key and send it to an authentication server. Further, it should be able to receive the authentication response from the authentication server and interpret it. On detecting a response from the server indicating the presence of a Rootkit, it should be able to take necessary action like 'beeping the PC speaker'.

We propose an approach based on trusted computing that uses the remote attestation mechanism to verify the authenticity of the code. Here, the Trusted Platform Module running on the machine gets a snapshot of the code running on it and sends this code to an authentication server. The authentication server has a code of the VM Rootkit which it uses to detect the presence of the malware on the code running on the machine.

We make the assumption here that the trusted authentication server is not infected and has the code of possible malwares and the TPM has the capabilities mentioned above. Given this assumption, we propose a protocol based on remote attestation to detect the presence of a VM Rootkit. It is based on the following steps.

5.1 Generating the authentication request

The Trusted Platform Module running on the suspected system gets a snapshot of all or part of (if it has an idea of the range of memory addresses where the Rootkit may be installed) code running on the machine. It then uses a hash function (the knowledge of which it shares with the authenticating server) generate the hash of the code for integrity. The TPM then encrypts this message consisting of the code and its hash with it's private (Endorsement) key sends it to an authentication server.

5.2 Client sends authentication request to authentication server

After having taken the snapshot of the code currently executing on the machine and generating its hash, the client encrypts this information, along with its ID (IP address), using the server's public key. It then sends this message to the authentication server using a dedicated port (part of TPM). Note that, by encrypting the message in the server's public key the client makes sure that no one except the intended server can observe the code running on the client. Otherwise, if the message was encrypted with the clients private key this would make it available to the attackers who can exploit this.

5.3 Authentication server checks authenticity and integrity of authentication request

The authentication server upon receiving the message decrypts it using the client's public key. It then retrieves the code and generates a hash of the code and compares it with the hash embedded in the message relieved. A matching hash value would insure the integrity of the message. The authentication server also compares the address in the source field of the message to the address of the client embedded in the message. Matching ID values indicate that the message is authentic. Otherwise, the server simply ignores the message considering it as a possible attack.

5.4 Authentication server looks for malware

To begin with, the trusted third party must have the code of the VM Rootkit. Apart from this, it may have some additional information about the VM Rootkit like the relative address mapping of the VM Rootkit that it may use for detecting the Rootkit.

Sliding Window based approach similar to the one used by Polygraph [7] is used to detect the malware. The authentication server has a mechanism by which it takes the code of the VM Rootkit and conceptually slides it over the code received from the client. The size of the sliding window should be equal to the size of the memory map of the code. The code is simply slid over the suspect code and matching bits are detected using AND logical operation. This problem is solvable in polynomial time. The time-complexity would be $\theta((m-n)*c1*c2)) \sim \theta(m-n)$ where 'm' is the size of the code running on the machine, 'n' is the size of the window (= size of the code) and c1 and c2 are constant time defining the time to perform AND operation and time to check for a match respectively. The time-complexity is $\theta(m-n)$ instead of $\theta(m)$ because, the algorithm always compares a piece of code that is exactly the size of code m. Thus, it will not compare the last n bits of the code.

Apart from this, we suggest an approach based on analyzing static code to detect its dynamic behavior though this may be difficult considering the large size of the code running on the machine. In particular, this approach may be used if; any part of the code running on the machine has similarities with the codes of the known Rootkits. It is to be noted that this method can be used to detect any kind of malware apart from VM Rootkit.

5.5 Authentication server sends authentication reply to client

If there is a match found to any of the malware codes using the above detection process, the authentication server sends back a message ATTACK indicating the presence of malware. Otherwise it sends a message OK indicating that the code was successfully authenticated ie. malware was note detected. The authentication server before sending this message adds its identity and encrypts it in its private key. Note that there is no need to hide this message from an attacker as it is merely a decision message.

5.6 Client receives authentication reply and performs necessary action

The client then receives the authentication response from the authentication server and decrypts it using the server's public key. It compares the identity information found in the message to the address of the intended server to insure its identity before proceeding. If there is a mismatch it ignores the message and waits for sometime for a valid reply. If there is no valid reply from the server side it times out and sends a new authentication request to the server. On detecting a positive response from the server indicating ATTACK, the client takes appropriate measures like *triggering a beep in the PC speaker* or *interrupting the display hardware to display a warning*.

6. Security Issues

It is very important to address these security issues to make the protocol robust. We classify the security issues as follows:

6.1 Rate for performing code authentication

What is the time-period for performing such a check? If the time interval is chosen to be too large the VM Rootkit may have already caused sufficient damage like stealing the private (Endorsement) key from the TPM or blocking the dedicated communication port. On the other hand, if this period is chosen to be small it may result in large computational and communication overhead; bringing down the performance of the system. However, it is not possible to arrive at an optimal period without practical experiments with Rootkits.

6.2. Trusted platform related Issues

There are various issues that need to be addressed while designing a Trusted Computing based platform for this system. Foremost, among them is to make the private (Endorsement) key secure and inaccessible. Next is the issue of building a Trusted Platform Module on top of 'root of trust' component that should perform the necessary operations. However, this design is not secure unless there is a mechanism to thwart the attacks on this system that can break its integrity. This paper [6] classifies the attacks on a Trusted Platform as follows.

6.2.1 Classification based on system component affected

6.2.1.1 Software attacks

The Root of Trust is usually implemented using some part of BIOS. So an attacker may try to modify this component of the BIOS to prevent the TPM from operating properly.

6.2.1.2 Hardware attacks

The communication among the components of Trusted Platform Module can be monitored to extract the private (Endorsement) key. This can be done using techniques

like ‘probing’ the BUS signals or using Side Channel Attacks (use information about the physical implementation of TPM)

6.2.1.3 Attacking the communication channel

TPM should have a dedicated port for communicating with the Authentication Server. The attacker may try to disrupt communication through this port thus disabling the machine from verifying the authenticity of code running on it.

6.2.2 Classification based on methodology of attack

6.2.2.1 Passive attacks

These types of attacks usually eavesdrop on the bus to monitor the activities of the Trusted Platform module. Once the attacker manages to read sufficient details about the TPM, he can use this information to duplicate TPMs. This breaks the notion of trust.

6.2.2.2 Active attacks

These types of attacks try to communicate with the TPM in an attempt to extract its configuration details. They may subsequently modify the operation of the TPM for example, cheating the Remote Attestation process.

6.3 Securing the Communication between the client and the authentication server

The server can verify the authenticity and integrity of the message using the ID (IP address) and the hash value embedded in the message. The identity is used to verify if the message was generated by the entity whose address appears in the Source field of the associated IP packets. The hash obtained by decrypting the message can be used to verify the integrity of the message as the hash is a one-way function i.e. a unique message would generate a unique hash. The server hashes the contents of the message received using the same hash function and compares this hash value with the hash embedded in message. If there is a match, this will insure the integrity of the message. Similarly the authenticity of the response sent by the server can be verified using the private key of the server in a fashion similar to that done at the server side.

We discuss some security attacks that can be launched on the system and ways to counter them below.

6.3.1 Replay Attacks

A Replay attack can occur when the messages sent by the authentication server are intercepted and then replayed back to the requesting client. Consider the scenario where at some point of time the authentication server did not find any malware in the code sent by the suspecting client and sends an OK message indicating the same. This message is intercepted by an attacker by some means like sniffing the packets and looking into the source and destination addresses for the address of the suspecting client and the

authentication server respectively. Now, say that the same attacker launches a VM Rootkit based attack. After managing to successfully drop a VM Rootkit in the machine, the attacker would obviously be interested in hiding it from the authentication server, which can detect the presence of the Rootkit on the code executing on a given machine using a pre-defined code. To accomplish this he first detects the next authentication message sent from the client to the server and then replays the OK message that it had intercepted some time back. This deceives the client into believing that there is no malware detected on the currently running code though, the machine has been infected by the VM Rootkit deployed by the attacker. We suggest using a NONCE or a TIME-STAMP to thwart a replay attempt by an attacker.

A unique NONCE can be embedded into the message that is encrypted by the private Endorsement key of the client machine. This message is then decrypted by the authentication server using the client's public key to the get the value of the Nonce. This NONCE value is then decremented (using any 1-1 function shared by the client and the server) by the server and then embedded into the reply sent by the server. If the client detects that the NONCE value sent by it matches the NONCE in the message from the server then it can be sure that this is not a replay attack. It is to be noted that both the client and the server have to maintain a buffer of recently used NONCE values to detect a replay attack. The NONCE embedded in the current message is compared with the previous NONCES. If the NONCE value matches any of the NONCE values available in the buffer the party can be sure that this is a reply attack as the NONCEs are unique.

A TIMESTAMP may be used to insure the freshness of the message and the two parties may simply drop older messages. Say that the two parties have a mechanism to find the propagation delay involved in the communication between the two parties (e.g. by measuring the round-trip-time involved with the PING message). Then, if either the client or the server detects a message that has a TIMESTAMP value for which the absolute difference between the TIMESTAMP and the receiving party's local time is greater than the propagation delay, then it simply drops the message. It is to be noted that this approach is vulnerable to attacks that target the modification of the local time on the machines. Thus a robust time-synchronization mechanism has to be build on top of time-synchronization protocols like NTP that can detect clock drifts (variation between local clocks on different machine) and skew (variation in the clock on a local machine).

6.3.2 Man-in-the middle attack

A Man-in-the-middle is a security attack where an attacker is able to intercept the communication messages between two parties and read, modify them or insert new messages. In our system, the attacker may modify any of the fields of the messages exchanged by the two parties at will since it can read the messages. It is based on making the client and server believe that the public key that they are using belongs to each other when it actually belongs to the attacker who can thus intercept and modify the messages from both side. This attack comes into picture only if it is successfully carried out during the initial (public) key exchange between the server and the client.

The attacker achieves this by reading the messages involved in the initial key exchange between the client and the server. The message containing the public keys is intercepted by the attacker and modified to provide different public key. For example, the attacker may modify the message containing the public key of the client and the server to replace it with his own public key. In case of the authentication request send from the client to the server, the message will now be encrypted using the public key of the attacker instead of the intended server.

This would allow the attacker to read the content of the message containing the code running on the machine. He may then modify the contents with a version of code that is known to be safe ie. it will not be labeled as malware by the authentication server. The attacker may know if a given code will be labeled as malware or not by sending a message to the server containing the code encrypted in his private key and observing the response from the server using the server's public key. Once he has determined a safe code, he encrypts this code using his private key or the server's public key or both depending on the procedure used. In the next step, he intercepts the reply message from the server, which is going to be OK, and then encrypts it using his public key before sending it to the client. As a result, the client is always duped into believing that there is no malware running on it

It is to be noted that, in this type of attack, the attacker has to make sure that he does not allow any direct communication between the client and server. Otherwise, the client or server may smell something suspicious if it detects two different messages with two different public keys intended for the same entity.

6.3.3. Brute-force attack

A Brute force attack is a security attack where the attacker may try to break the encryption mechanism by trial and error using cipher text or plain text-cipher text pairs. This may be used by the attacker to gain access to the client's private (Endorsement) key. Once, the attacker gains access to the private key embedded in the Trusted Platform Module on the client, he has broken the trust placed in the trusted platform module. This is equivalent to saying that is not trusted computing anymore.

An attacker may gain access to the cipher-text by using techniques like packet sniffing. Once the attacker has a set of cipher texts, he may try to use different encryption algorithms and keys to decrypt the message to see if the text thus decrypted makes any linguistic/semantic sense. Further, if he manages to get access to the cipher text-plain text pairs, then this makes his task easier by giving him more to compare with. In this case, he can take a given plaintext and try to apply different encryption algorithms with different keys to see if it produces the corresponding cipher text.

6.3.4 Denial-of-Service attacks

A denial-of-service attack is a security attack that targets the availability of the system itself. In the case of our system, the attacker will try to make the authentication system

unavailable. The attacker can achieve this by thwarting any attempt by the client to authenticate the code running on it by communicating with the authentication server. There are many possible scenarios for launching such an attack.

The simplest scenario is where the attacker intercepts or blocks all the authentication requests sent from the client to the authentication server, thus preventing any such request from reaching the authentication server. It may be do this using some security tools installed on the communication path for example in the routers and switches.

Other way to do this would be to flood the authentication server with a lot of requests which may cause traffic congestion in the network. This will prevent the authentication server from authenticating the client's code at periodic intervals thus giving a better opportunity for the Rootkit to infest the system before it can be detected. As discussed in section 6.1, this may enable the Rootkit to gain more control over the system like blocking the channel used for authentication purposes.

6.4 Miscellaneous Issues

There is the initial key exchange problem where the client and server exchange each others public key to be used for subsequent communication between the two parties. For this we suggest in-personal exchange of keys for safety. Robust security mechanisms have to be used to secure the authentication server that is an important part of this protocol.

7. Related Work

This paper[1] analyzes this topic of implementing and detecting malware with Virtual machines. The primary argument put forward by this paper is that the security components have to be present at a lower abstract logical level than the malware itself to be of reliable use. They implement two different VM Rootkits called 'Subvirt' for Windows XP and Linux target systems. They discuss the design and implementation of VM Rootkits in detail. They also discuss the various malicious services that can be implemented using such a malware. These services are hidden from the native OS (that runs as a Guest OS) by running a different OS called Attack OS. The malicious services are classified into three categories. First kind of services that that need not interact with the target OS, are run on other OS called Attack OS like spam relays, distributed denial of service, zombies and phishing web-server. Second type of service observes data or events from the target system like the hardware level data which includes keystroke logger and network packet monitor. Third type of service hinders the execution of the system e.g. modifying network communication, delete e-mail messages etc. They also discuss some detection techniques for this type of malware.

They classify the detection techniques based on whether the security software runs below or above the VM Rootkit. The security software that runs below the VM Rootkit is more effective in detecting the Rootkit by means of reading the physical memory and looking

for anomalies. They suggest running low level security software that runs below the VM Rootkit. They stress on getting the boot sequence right though this seems a bit outdated as recently developed VM Rootkits like Blue Pill and Vitriol can install themselves even during the runtime and do not require a modification of the boot sequence.

This paper [5] explores ways to tackle the VM Rootkits that work by changing the boot-sequence. It implements a prototype called the BootSafe that prevents the Boot Firmware from being changed. They use Efficient Code Certification technique that performs static checks on the relevant code to predict dynamic properties of the code.

This paper discusses the design and implementation of a TCG-based Integrity Measurement Architecture for Linux [10]. All executable content that is loaded into Linux, is measured for integrity before it is run. Trusted Platform Module (TPM) is used to protect all these measurements. It uses the TCG trust measurement concept from the BIOS all the way to the application layer.

IBM research has developed a Virtual Trusted Platform Module [7]. This platform is used to implement a trusted platform on multiple Operating Systems running on the same physical platform. They have added additional instructions to the TPM 1.2 for this purpose. They also support interesting features like migrating the current configuration of the TPM along with the Virtual Machine. This includes the migration of storage keys among the systems sharing them.

In a slightly different note this paper[6] proposes a 'Virtual Machine based platform for trusted computing'. Terra uses a Trusted Virtual Machine Monitor (TVMM) that partitions a tamper-resistant trusted platform into multiple isolated virtual machines. For each of the VMs, the TVMM provides the semantics of either a open-box or closed-box hardware platform. Open-box hardware platform is a general purpose hardware platform like today's PC whereas Closed-box is an opaque and special purpose platform that protects privacy and integrity of its contents.

8. Conclusions and Future Work

Thus we propose a Trusted Computing based Remote Attestation protocol to detect VM Rootkit running on a system. Then we evaluate the protocol from security standpoint by analyzing different attack techniques and ways to get over them. We also analyze other approaches that can be used to detect VM. We think that the Trusted Computing approach can get over the problem of false positives and negatives.

Other efficient techniques can be explored for the detection of malware in a given code. Vendors may consider adding a security module along with the TPM running at hardware level in future.

References

1. Samuel King, Peter M. Chen, Yi-Min Wang, Chad Verbowski, Helen J. Wang, Jacob R. Lorch, SubVirt: Implementing malware with virtual machines, *Proceedings of the 2006 IEEE Symposium on Security and Privacy* , May 2006

2. Joanna Rutkowska, Subvering VISTA Kernel for Fun and Profit, SyScan '06 and Black Hat briefing '06

3. Joanna Rutkowska, Introducing Stealth Malware Taxonomy, COSEINC Advanced Malware Labs, Nov. 2006

4. Reiner Sailer, Xiaolan Zhang, Trent Jaeger and Leendert van Doorn, Design and implementation of a TCG-based Integrity Measurement Architecture, Proceedings of he 13th USENIX Security Symposium, August 2004

5. Frank Adelstein, Mat Stillerman, Dexter Kozen, Malicious code Detection for Open Firmware, Proceedings of the 18th Annual Computer Security Applications Conference 2002.

6. Klaus Kursawe, Dries Schellekens, Bart Preneel, Analyzing Trusted Platform Communication, Katholieke Universiteit Leuven, Belgium, July 2005

7. Newsome, J., Karp, B., and Song, D., Polygraph: Automatically Generating Signatures for Polymorphic Worms, *Proceedings of the IEEE Symposium on Security and Privacy*, May, 2005

8. Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, Dan Boneh, Terra: A Virtual Machine-Based Platform for Trusted Computing, SOSP October 2003

9. Stefan Berger, Ram´on C´aceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, Leendert van Doorn, vTPM: Virtualizing the Trusted Platform Module, 15th USENIX Security Symposium, 2006

10. K. Hwang, Y. Chen, and H. Liu, "Defending Distributed Systems Against Malicious Intrusions and Network Anomalies," Keynote Presentation by Kai Hwang in the IEEE International Workshop on Security in Systems and Networks (SSN'05), in conjunction with the IPDPS-2005, USA. April 8, 2005

11. Keith Adams, Ole Agesen, A Comparison of Software and Hardware Techniques for x86 Virtualisaion, ASPLOS, October 2006

12. Trusted Computing Group, TCG Specification Architecture Overview, Revision 1.2, April 2004