Towards a Trusted Immutable Kernel Extension (TIKE) for Self-Healing Systems: a Virtual Machine Approach

Julian B. Grizzard, Eric R. Dodson, Gregory J. Conti, John G. Levine, Henry L. Owen
Georgia Institute of Technology; Atlanta, Georgia 30332–0250
Phone: 404.894.2955 Fax: 404.894.9959 Email: grizzard@ece.gatech.edu (presenter)

Abstract—

The conventional method to restore a compromised system is to wipe the system clean, install from known good media, and patch with the latest updates: a costly, restrictive, and inefficient method. An alternative method is to monitor the host and restore trust if a compromise occurs. When this method is automated, the system is said to be *self-healing*. One critical requirement of a self-healing system is that the self-healing mechanism itself must not be compromised. Our solution to this requirement is a Trusted Immutable Kernel Extension (TIKE) by way of a virtual machine. Using a host operating system as a trusted platform, we discuss a self-healing system that uses existing intrusion detection systems and corresponding self-healing mechanisms to automatically heal the guest operating system once a compromise has occurred.

I. Overview

The conventional method to recover from a system compromise is to wipe the system clean and perform a fresh installation. As an alternative, it has been suggested that computers can model the human immune system [1]. Our method is to automatically re-establish trust in the compromised system or to build a *self-healing* system. However, in most existing systems the entire state, including the kernel, can be altered to an untrusted state once an attacker has gained *root-level* privileges [2]. Hence, even the self-healing mechanism itself could be compromised. In order to further explore the notion of self-healing systems, a core foundation of trust is needed. Our solution is a Trusted Immutable Kernel Extension (TIKE), and we explore this concept with a virtual machine approach. TIKE can be used as a safe haven for self-monitoring and self-healing a system. Eventually this work can be extended to the distributed environment to extend the work of Ostrovsky and Yung [3] so that an entire computer network can model the human immune system.

Figure 1 shows an overview of the TIKE Architecture. The guest system is considered to be the *production system*. The *guest applications* include common user applications such as email programs, web browsers, system tools, and so forth. The *TIKE applications* are the tools that monitor and repair the guest system. The TIKE applications run in the host system and are therefore *isolated* from the guest system. We assume the integrity of the host machine will not be compromised. Therefore, the host system should be completely transparent, isolated, and inaccessible from inside the guest system. Terra is an example of other work that uses virtual machines for isolation and security [4].

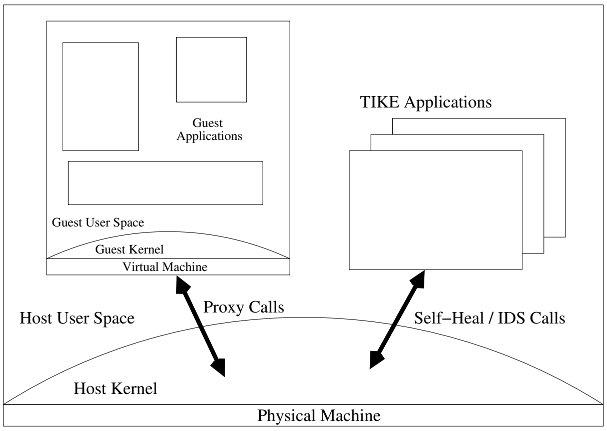

Fig. 1. Overview of TIKE Architecture

II. Design Principles

A. TIKE Requirements

The requirements of TIKE are embedded in its name, a Trusted Immutable Kernel Extension. Below, we describe the three core requirements of TIKE.

- *Trusted* – TIKE must report accurate information about the state of a host, which means the information can be trusted to be true. Furthermore, in the realm of self-healing, TIKE must be trusted to correctly self-heal the system.
- *Immutable* – In order for TIKE to be trusted, it must be immutable. If an attacker compromises a system, the attacker must not be able to compromise TIKE. Further, the attacker must not be able to disable TIKE's services.
- *Kernel Extension* – In order to monitor the entire state of the system, TIKE must exist at the kernel level.

In addition to the requirements listed above, TIKE must have the capability to examine and modify any state within the host. Furthermore, TIKE should have a small impact on system performance that is not noticeable to the user under normal operations.

B. TIKE Virtual Machine Architecture and Operation

The TIKE virtual machine architecture consists of a *host* operating system and a *guest* operating system. The host operating system is considered the core element of trust that is immutable, or the Trusted Immutable Kernel Extension. Normal users (even those with root access) are *not* provided access to the host operating system. The host operating system has complete visibility of the entire guest system. Normal users will have accounts with the required level of access on the guest operating system.

The host operating system boots up on the physical hardware. After the host operating system is loaded, it then loads the guest operating system. Once the guest operating system is running, TIKE applications can be launched on the host operating system to monitor, repair, or otherwise control the guest operating system.

One of the key ideas to note is that there is a small well defined interface between the guest operating system and the host operating system. From a remote location, there is *no* other interface to the host operating system (i.e. the host O.S. is isolated from the network). The problem of proving whether or not it is possible for the host operating system (i.e. TIKE) to be compromised is thus greatly reduced to proving that a very simple interface is correct and that the TIKE applications are correct, which is arguably more feasible than proving that an entire operating system is correct.

III. Self-Monitoring and Self-Healing

Using the TIKE architecture, self-monitoring and self-healing mechanisms can be installed on the host operating system, isolated from the guest or *production* system. These mechanisms can monitor the production system and repair or *heal* any compromises that occur. Existing intrusion detection systems can be used for both real-time and post-intrusion analysis.

One of the difficulties with this architecture is that visibility inside the guest operating system is limited. It is possible to see the entire state of the guest operating system from the host operating system but some complexities exists. For example, it is possible to read the guest file system but it proves difficult to read I/O caches since this may require parsing the guest operating system's kernel data structures. We will continue to look at solutions to this problem.

IV. Limitations

The TIKE architecture we have described is a step forward towards building a Trusted Immutable Kernel Extension. However there are some limitations that we would like to note. First, one assumption that we make is that there are no vulnerabilities in the layer between the guest operating system and the host operating system. If this assumption is false, then the whole model dissolves. However, our argument is that this interface is sufficiently simple, which should enable substantially easier verifiability than say a full blown operating system. Another limitation is performance. With a virtual machine architecture, system performance will be effected. However, the argument is that many production systems in use today and the foreseeable future have far more power than is required. A final limitation that we will point out is that this architecture alone offers no protection from physical security. The architecture focuses on protecting systems from remote attackers. A local attacker could, for instance, boot the system from a cdrom or other bootable medium and obtain full control over the system.

V. Closing and Future Work

We have provided an overview for a Trusted Immutable Kernel Extension by way of a virtual machine. The guest operating system is considered the *production* system and the host operating system is considered TIKE. The TIKE architecture is an enabling architecture that can provide self-monitoring and self-healing mechanisms with a safe haven so that the mechanisms themselves are not compromised. Future work will include a more rigorous examination of the interface between guest operating system and host operating system, methods to increase visibility inside the guest operating system, guest kernel self-healing mechanisms, and methods to withstand local attacks. Finally, we realize that other architectures exists other than the virtual machine approach that meet the requirements of TIKE, such as implementing a hardware module.

VI. Acknowledgments

The authors would like to thank Dr. Karsten Schwan for his valuable guidance and direction, Ivan Ganev for his suggestions, Dr. Wenke Lee for his insights into intrusion detection systems, and Dr. Mustaque Ahamad for his critical review and comments on trusted computing systems. We would also like to extend our thanks to the anonymous reviewers for their valuable comments.

References

[1] S. Forrest, S. A. Hofmeyr, and A. Somayaji, "Computer Immunology," *Commun. ACM*, vol. 40, no. 10, pp. 88–96, 1997.

[2] J. Levine, J. Grizzard, and H. Owen, "A Methodology to Detect and Characterize Kernel Level Rootkit Exploits Involving Redirection of the System Call Table," in *2nd IEEE International Information Assurance Workshop*, pp. 107–125, IEEE, 2004.

[3] R. Ostrovsky and M. Yung, "How to Withstand Mobile Virus Attacks (extended abstract)," in *Proceedings of the tenth Annual ACM Symposium on Principles of Distributed Computing*, pp. 51–59, ACM Press, 1991.

[4] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: a Virtual Machine-Based Platform for Trusted Computing," in *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles*, pp. 193–206, ACM Press, 2003.