Kernel Framework for an Immune-Based Security System: A Work-In-Progress Report

Martim d'Orey Posser de Andrade Carbone[1]*, **Paulo Lício de Geus**[1]

[1]Institute of Computing – State University of Campinas (UNICAMP)
PO Box 6176 – 13083-970 Campinas, SP

{martim,paulo}@las.ic.unicamp.br

Abstract. *This report informs on the current status of the project whose goal is to design, implement and integrate into the 2.6 version of the Linux kernel a generic framework to support a computer security system inspired in the principles of the human immune system. A brief introduction to the project is given, followed by a more in-depth discussion of the framework requirements and its overall architecture. It concludes by pointing out the future research and development stages.*

1. Introduction

Modern security threats create the demand for robust and complete models on which security systems can be built upon, such as the human immune system. This system is capable of automatically recognizing and responding to a wide variety of biological and chemical threats, as well as to learn from those threats and accelerate future responses. It successfully integrates prevention, detection, and reaction counter-measures in a fully adaptative and automatic manner, making it extremely desirable for a computer.

Fundamental steps towards the construction of such a system were made before by former lab members working at the *Imuno* project [de Paula et al. 2004, de Paula et al. 2002]. In this project, we aim to continue their work by implementing a framework that will allow the integration of that system into the Linux operating system. This framework will consist mainly of a set of hooks (alongside with its access and management infrastructure) implemented inside the Linux kernel (version 2.6) to address the low-level prevention, detection, learning and response needs of the immune modules.

The framework will be implemented as an extension of LSM (*Linux Security Modules*) [Wright et al. 2002], an established framework for granular access control support inside the Linux kernel. The main goal is to generalize LSM's access control infrastructure by creating a more sophisticated and dynamic hook management engine, so as to support other security functionalities, and also complement it with additional hook points. The CKRM (Class-based Kernel Resource Management) [Nagar et al. 2004] and Netfilter [Russell and Welte 2002] frameworks will also be used in this project.

This paper is organized as follows: it will first discuss the framework's requirements in Section 2., followed by the first model of its overall architecture, in Section 3.. Finally, Section 4. will conclude with some closing remarks and future steps.

*Supported by CNPq

2. Framework Requirements

As stated in Section 1., the framework must support all major security mechanism classes in order to accommodate a complete artificial immune system. This includes prevention, detection and response mechanisms, with two additional classes that are particularly important to this project: self-protection and learning. Requirements are listed below. More detailed explanation and specific requirements were omitted due to space constraints.

- **Prevention**: The framework must provide support for complete, granular resource access control and must be generic enough so that multiple access control models may be implemented on top of it. The LSM framework already provides a good implementation of these functionalities, so it will be used as a foundation for access control in this framework;
- **Detection**: The framework must provide multiple in-kernel sources of detection such as system call execution, on and off-memory disk blocks, signals, IPC messages, memory pages, and network packets (which is where Netfilter fits). These sources must also be accessible in an efficient manner by the detection modules, which will reside in userspace;
- **Learning**: The framework must support the maintenance of a forensic knowledge base, containing information concerning intrusion signatures, system activity profiles and response measures. It must also allow forensic modules to look back into parts of the system activity history and its present state, so that the analysis has plenty of data to work with.
- **Response**: The framework must allow two types of automated response:
  - **General**, non-damaging response, used in the early stage of the response process. Its goal is to keep the operating system basic functions online, in spite of a possible intrusion situation, and without causing any permanent, potentially damaging changes. The framework must support several response techniques such as resource control (disk, memory, processor), system call delaying, process control and scheduler policy control. This component will use parts of CKRM [Nagar et al. 2004], a work-in-progress framework for resource management inside the Linux kernel.
  - **Specific**, targeted response, activated after the threats have been identified and a proper response has been planned for their elimination. This includes active measures such as file removal, connection closing, process restoration and filesystem restoration.
- **Self-protection**: The framework must provide mechanisms to ensure that itself and the immune system modules stay protected from attacks, even if the attackers have acquired *root* privileges on the system.

3. Overall Architecture

The artificial immune system is divided into a userspace and a kernelspace portion. The userspace portion includes all immune modules, responsible for dictating the policies and executing the actual algorithms that control the system's activities. The kernelspace portion is the framework that has been discussed so far. Modules were chosen to reside in userspace because of improved mathematical support through libraries (which is important for some algorithms), IPC support and the possibility of using programming languages other than C. None of these conveniences are available in kernelspace.

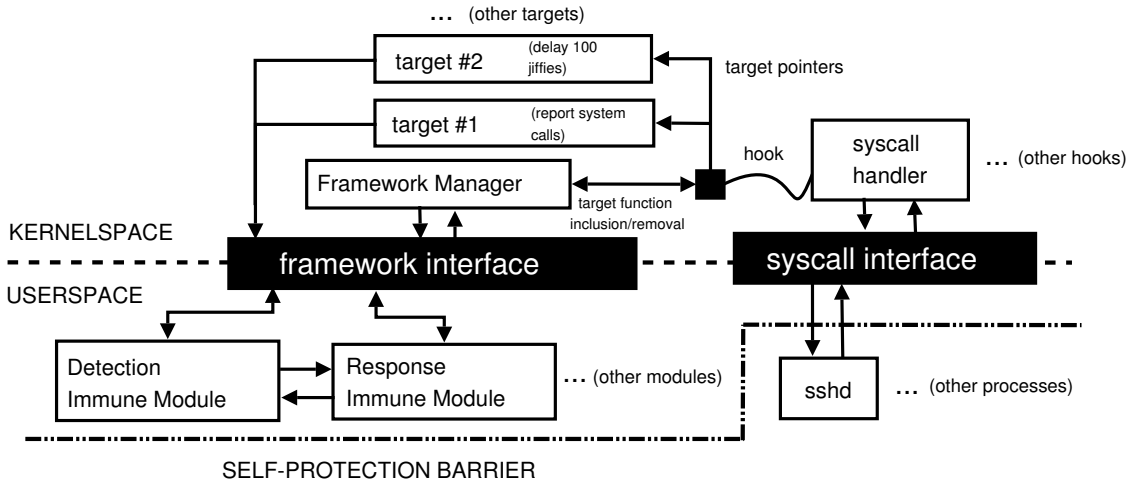

Figure 1. Overall framework architecture

The framework basically consists of various in-kernel hooks. These are generic invocation commands that point to certain target functions implemented by external modules. Classical hooks execute them and act based on what these functions return. In this framework, hooks are the means through which the immune modules will actually control the prevention, detection and response activities of the immune system.

Nevertheless, the hook architecture proposed here is much more sophisticated than the one used in frameworks such as LSM. In LSM, hooks have a purely restrictive nature, that is, they simply allow or disallow a certain action to be performed based on what a target function returns. Moreover, hooks are used statically, which means that their target pointers are not designed to be changed very often, in a real-time basis. And finally, LSM-based modules reside in kernelspace, which makes management much simpler. The architecture proposed here expands LSM's, making it support multiple targets by hook and a dynamic target inclusion/exclusion mechanism in which a single hook can perform a wide variety of activities and its targets be changed in real-time. The hook targets are uploaded from userspace by the immune modules using a framework interface. The key elements of this architecture are portrayed in Figure 1's example.

It illustrates one of the framework's hooks, placed at the system call handler function. Classically, this hook would be used only for a unique and specific purpose such as reporting system call executions. In this architecture, however, it can perform a myriad of activities, being a truly multi-functional hook. For example, an intrusion detection module may upload from userspace a target function that reports system call executions, in order to monitor the behavior of one or more user processes (*sshd*, in Figure 1). This target is attached to the hook by the framework manager. Simultaneously, a response module may want to use that same hook to command all *open()* system call executions to be delayed by 100 jiffies[1]. The targets are executed sequentially. The response module may, at a certain point, decide that it wants to increase the delay by 50 jiffies, so it commands the manager to modify the time delay parameter, in real time. And finally, it may decide to disable delaying, so it commands the manager to delete that particular target function. Target functions

[1]A jiffy is the unit of time equal to one clock tick of the system's timer.

may also send data back to the modules through the framework's interface.

Figure 1 also depicts a self-protection barrier that isolates the immune system components from the rest of the system. Since it is hard to create a robust implementation of this requirement working with a non-trusted computer architecture, our implementation will follow the simplest and most effective software-based alternative, which consists of kernel memory isolation. This implicates in disabling common interfaces between user-mode and kernel-mode such as module loading and special device files, like `/dev/kmem`. Once the kernel is protected, it then becomes a matter of extending this protection to the userspace immune modules. This policy can be enforced through LSM.

4. Conclusion and Future Steps

This report briefly introduced and summarized the main steps that have been fully or partially concluded in our work towards the implementation of a kernel framework for an immune-based security system. These steps included the framework requirements list and a first model of its overall architecture.

Future steps include the detailing of the architecture model, particularly the precise definition of where the hooks will be placed; and subsequent implementation and integration into the Linux kernel. We plan to make use of several established design and implementation techniques already being used in other open-source kernel security projects. Finally, we plan to have the framework tested for correctness and efficiency by using real and/or stub immune modules and performance diagnostic programs such as *LMBench*[2].

It should also be noticed that the broad definition and generic architecture of our framework should allow it to support not only an artificial immune system, but also a wide variety of security applications such as access controllers, host-based IDSs, IPSs, forensic tools, antivirus software, firewalls, resource control tools, among others, thus making it a truly generic kernel security framework.

References

de Paula, F. S., de Castro, L. N., and de Geus, P. L. (2004). An Intrusion Detection System Using Ideas from the Immune System. In *Proceedings of the IEEE Congress on Evolucionary Computation*, Portland, Oregon, USA.

de Paula, F. S., dos Reis, M. A., de Assis Monteiro Fernandes, D., and de Geus, P. L. (2002). ADENOIDS: A Hybrid IDS Based on the Immune System. In *Proceedings of the 9th International Conference on Neural Information Processing*, Singapore.

Nagar, S., van Riel, R., Franke, H., Seetharaman, C., Kashyap, V., and Zheng, H. (2004). Improving linux resource control using CKRM. In *Proceedings of the 2004 Ottawa Linux Symposium*.

Russell, R. and Welte, H. (2002). Linux netfilter hacking HOWTO. Disponível em http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO.letter.ps (Junho de 2005).

Wright, C., Cowan, C., Smalley, S., Morris, J., and Kroah-Hartman, G. (2002). Linux Security Modules: General Security Support for the Linux Kernel. In *Proceedings of the 11th USENIX Security Symposium*.

[2]http://www.bitmover.com/lmbench