

XenAccess: An Introspection Library for Xen

Bryan D. Payne
College of Computing
Georgia Institute of Technology
bdpayne@cc.gatech.edu

February 5, 2006

1 Introduction

Mounting a successful defense against malicious software is growing more difficult each day. Sophisticated attackers can subvert nearly any process an administrator runs on a machine – in user space and even in the kernel – using techniques such as process hijacking, kernel rootkits, and software injection. In light of this type of advanced threat, typical host-based security tools have little value because their state and output can be adjusted to meet the attacker’s goals. Furthermore, at a broader level, under this threat model system management tools quickly break down and prevent administrators from maintaining control of their assets.

Recent research has considered a variety of techniques to defend against these types of threats. However the solutions thus far have either required adding additional hardware to the machine [PFMA04] or modifying the user’s operating system to add robust security features [SLS⁺05]. Adding additional hardware can be costly and only provides a limited view into the running system. Likewise, modifying the user’s operating system can negatively impact application availability, user productivity, and administration cost.

One promising alternative is a technique known as virtual machine introspection [GR03]. Introspection is a technique used in conjunction with a hypervisor and multiple virtual machines. Under this configuration, a virtual machine can be granted specific privileges by the hypervisor to monitor the other virtual machine(s). The monitoring provides a view into an operating system’s memory, disk and CPU information from an isolated and protected environment.

For the course special project, I propose building an introspection library for use with the Xen hypervisor. The library, known as XenAccess, will provide the privileged domain simplified access to the systems running in the guest domains. The remainder of this proposal describes the scope and plan for this project.

2 Project Approach

The Xen hypervisor provides the basic functionality needed to implement the XenAccess library. Therefore, this project will not focus on modifications to the Xen hypervisor or its related libraries. Instead, this project will focus on using the features in Xen to facilitate introspection. In general, the primary benefit that the XenAccess library will provide is a higher-level abstraction than what is currently available through Xen’s libxc library. For example, the libxc library allows a privileged domain to map any part of memory using the machine frame number whereas the XenAccess library will allow the same privileged domain to access another domain’s system call table directly.

To further motivate this example, Program 2.1 shows how one could use the XenAccess library to monitor another machine’s system call table: The XenAccess library functions in this example all start with the prefix `xa_`. The string “`sys_call_table`” is a reference to the symbol in the `System.map` file associated with a Linux kernel.

Development work on the XenAccess library has already begun, but it is in the very early stages. The existing work is sufficient to show that the technique is possible. In order to understand what has been completed already, we need to first provide a high level overview of the memory layout in Linux. Linux divides its virtual address space into two major

Program 2.1 The XenAccess Library provides simplified access to a guest domain's system call table.

```

xa_init();
memory = xa_access_kernel_symbol("sys_call_table");
while (1){
    sleep(SLEEP_INTERVAL);
    if (memcmp(memory, known_good_syscall_table) != 0){
        /* syscall table has changed! */
    }
}
munmap(memory);
xa_destroy();

```

sections: user memory and kernel memory. User memory is positioned in the first 3GB of the virtual address space and kernel memory occupies the final 1GB¹. Within the kernel memory, there are three major regions:

- `0xC0000000` \rightarrow `high_memory`: This region is linearly mapped so it is trivial to locate the physical address of any memory in this region. It holds the kernel text and data regions.
- `high_memory` \rightarrow `0xFE000000`: This region is accessed with page tables. It holds dynamically allocated kernel memory, including module text.
- `0xFE000000` \rightarrow `0xFFFFFFFF`: This region holds some of the specialized kernel memory areas such as fixed-mapped linear addresses, permanent kernel mappings, temporary kernel mappings, and noncontiguous memory allocation.

The current proof-of-concept library can only view memory in the `0xC0000000` \rightarrow `high_memory` region. However, the proposed XenAccess project will extend this to include the rest of kernel memory and user-space memory. In addition, if time permits, the project will also add support for view disk and CPU information.

The primary means of experimentation and evaluation for this type of work is micro-benchmarking. Micro-benchmarking will be used to show how the use of introspection impacts system performance. For each type of introspection, a specific benchmark will be used to show how the technique impacts system performance. For example, for memory, the STREAM benchmark can be used.

3 Expected Outcomes

The expected outcome from this project is a robust introspection library that can be used to facilitate a variety of additional research. Virtual machine introspection is a powerful technique that can be applied to security monitoring & response applications and systems management research. Therefore, the XenAccess library will be designed with the idea of facilitating this type of research.

4 Schedule and Milestones

The project will be completed in three steps as outlined below:

- **Step 1 – Due March 10, 2006:** Complete the implementation of access to all kernel memory and develop a bibliography complete with abstracts of all papers relevant to this project. *Deliverables for this step include a snapshot of the software and a bibliography.*
- **Step 2 – Due April 7, 2006:** Complete the implementation of access to memory (kernel and user) and provide some facility for access to disk and CPU information. The level of access provided to disk and CPU information will depend

¹A recent patch submitted to the Linux kernel will provide a compile time option to change where the user / kernel partition exists. However, for purposes of discussion, we will focus on the traditional 3/1 split that is seen in most Linux kernels today.

on how rapidly the development progressed for access to memory. *Deliverables for this step include a snapshot of the software with all features tested and example code for using the XenAccess Library.*

- **Step 3 – Due Apr 28, 2006:** Complete more extensive software testing and benchmarking. Also complete the final report following the outline provided in the course project page. *Deliverables for this step include a “1.0” release of the software and the final report.*

In addition to the specific deliverables outlined above, a project web page will be created that will include frequent status reports, access to the software, and additional details about the project execution.

References

- [GR03] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the Network and Distributed Systems Security Symposium*, February 2003.
- [PFMA04] Nick L. Petroni, Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot – a coprocessor-based kernel runtime integrity monitor. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
- [SLS⁺05] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–15, Oct 2005.