An Investigation of the Call Integrity of the Linux System

Dayle G. Majors
University of Missouri - Rolla

Abstract— **The internal calls of a version of the Linux operating system were evaluated with the assistance of ITS4 source code scanner to identify security exposures such as buffer overflows. Some vulnerabilities were identified and some suggestions for improvements made.**

I. Introduction

The research discussed here was performed at the University of Missouri Rolla under the guidance of Dr. Ann Miller, Tang Professor Computer Engineering, with input from Dr. Bruce McMillin, Professor Computer Science. The Linux kernel calls were analyzed to determine if they presented the potential for exploitable security vulnerabilities. A version of Linux had been certified under the *Department of Defense Trusted Computer System Evaluation Criteria*[1], also known as the DOD Orange Book. However, an evaluation of a generally available commercial version was desirable.

The Orange Book stated, as a point of concern for security and integrity, that calls made by users should be evaluated. It specified what should or should not happen. The calls internal to the operating system were used to accomplish the task specified by the external calls. Very infrequently could a task be accomplished without calling other functions. Calls passed data to the called routine and usually expected a response. If the passed data were larger than the called routine expected, the called routine could overflow a buffer.

II. Kernel Analysis

Calls could be identified easily but the volume would be very large. To assist in the analysis, all of the Linux kernel data was processed by a package developed by John Viega and Gary McGraw called *It's the Software Stupid Security Scanner*, abbreviated ITS4. *ITS4: A Static Vulnerability Scanner for C and C++*[2] discussed the package. ITS4 contained a list of calls that were known to be potential risks in C or C++. The printk function was added to this list since its parameters were essentially the same as the printf function. This reduced the number of calls that would need to be examined manually.

For each call to a function in ITS4's list of functions with integrity concerns, the call was assigned a severity based on the call parameters. This severity reflects how much potential existed for the suspected vulnerability to cause problems. Only the higher severity calls were examined. The list of functions included file open functions, random number functions, task initiation functions and string processing functions including printf and printk. The kernel code contained few file opens but many device opens. Device opens did not have the same exposure to be exploited as file opens. Most of the calls highlighted by ITS4 involved variable length string processing.

III. Observations

In evaluating the highlighted calls, it was apparent that the Linux developers were not significantly concerned with ensuring their code did not have exploitable security vulnerabilities. Certainly kernel code should be coded so as not to have performance problems, but there is a fine balance that must be maintained between performance and integrity. They frequently did not verify the length of a string before calling sprintf. Such calls could modify the data beyond the receiving buffer. This would be a buffer overflow. By using snprintf instead of sprintf such modifications could be prevented. If that is perceived as a performance problem then perhaps snprintf should be improved, or it least shown not to have significant performance penalties.

Another example of a potential security vulnerability was a kernel functions that used the memcpy function to move parameter data from the application area to kernel area. To accomplish this move, the kernel function used the length of the data in the application area to allocate space in the kernel area. The memcpy function then used the length of the receiving area to move the data. That receiving area was passed to another function as a parameter. If this called program did not expect an input that large, then the called program could fail.

IV. Conclusions

There were about 3 percent of the kernel calls, excluding device drivers, which contained variable length strings as parameters where the function had not insured that the call would not overflow a buffer. Most of these calls could be fixed. Whether all have been fixed in the certified Fermi RedHat v7.3.1 version of Linux is not known.

Acknowledgement

This research was partially supported by a Graduate Assistantship in Areas of National Need (GAANN) from the United States Department of Education.

References

[1] *Department of Defense Trusted Computer System Evaluation Criteria*, United States National Security Agency, Washington, District of Columbia, Dec. 1985.

[2] J. Viega, J. T. Bloch, T. Kohno, and G. McGraw, "ITS4: A static vulnerability scanner for C and C++ code," in *16th Annual Computer Security Applications Conference*. ACM, Dec. 2000. [Online]. Available: ftp://ftp.rstcorp.com/pub/papers/its4.pdf