JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 5, May-June 2004

Book Review

Exploiting Software: How to Break Code

by Greg Hoglund and Gary McGraw, Addison-Wesley, Boston, MA, 2004. 448 pp., $39.99(paper). ISBN 0201786958.

Reviewed by Charles Ashbacher

To be useful, software must respond to events in a predictable manner. The results can then be used as a window to the interior workings of the code, revealing some of the mechanisms of operations, which may be used to find ways to make it fail in a dangerous way. To some, the window is as clear as a six inch thick pane of lead, but to those with a high level of understanding it can be clear, or at the very least serve as a keyhole. This is an allusion to the old detective stories where someone looks through the keyhole to see what is behind the door. For these reasons, no software that interacts with humans can ever be considered completely secure, and human error in the development of the software can leave the equivalent of keyholes throughout the code.

This book is an explanation of many of the most frequently used attack strategies used by malicious entities to find security flaws in code and exploit them. Chapter two is a list of the most common patterns used in attacking code, and all types of programs, from applications to compilers to network software are examined. In chapter three, the fundamental steps of reverse engineering source code starting with the executable are described in detail. I have had students who work in industry who have argued vehemently that it is not possible to obtain source code from executable. I knew it was possible, but until I read this chapter, I had no idea it was so easy. If you are releasing your programs as executables created directly from the source code, the examples here will very quickly make you reconsider. Without a doubt, you will be convinced that you should perform some form of obfuscation of the source before compiling or perform some type of encryption.

Chapters four and five are how to exploit server and client software respectively. From the perspective of the server, every input should be considered suspect, and you cannot assume that any scripting code embedded in the file was run at the client. In many cases, assumptions like this can create problems. People embed hidden fields or Javascript in HTML files and assume that the inputs are then clean, forgetting that all

Cite this book review as follows: Charles Ashbacher: Review of “Exploiting Software: How to Break Code”, in *Journal of Object Technology*, vol. 3, no. 5, May-June 2004, pp. 151-152. http://www.jot.fm/books/review12

such code is visible to a potential attacker. This is actually worse than nothing, because an attacker can look at the features and get a good idea about what it is you are afraid of receiving. Each chapter has a list of specific strategies that are used in attacks.

In chapter six, you get a very brutal lesson in the wisdom of filtering input and never forgetting that characters come in more than one form. Characters such as the slash and backslash are used in representing directory structures. Some code will filter them out, but fail to catch instances where they are sent in their numeric ASCII or Unicode form. One of the classic attempts to beat the filtering is to try the sequence "\/", in the hopes that the first will be considered an escape character, so that the slash can be embedded in a string. If that happens, then the slash could be used in a pathname. Many other possibilities exist to send code that is clearly malicious, but only if it is interpreted the proper way.

Chapter six is a complete tour of the most common security weakness found in software, the buffer overflow. It is the simplest problem to understand and one of the most difficult to remove. Every C programmer has had to find and repair a bug due to an off-by-one error, or some other overflow. And yet, despite all this experience, buffer overflows still are prevalent in commercial code. Most of the obvious ones have been removed, so only the very subtle ones remain. Some of these are very hard and very, very subtle. I was amazed in reading the section on format string vulnerabilities. While this bug has largely been repaired, the fact that something as apparently trivial as a format field specifier can be a security problem was a real eye opener.

The last chapter was an explanation of rootkits, the software that controls every aspect of the machine. It was also without question the scariest of all the chapters, because in this case, the malicious code could reside in the BIOS, and be largely immune to virus scanning tools. For the first time, we are talking about hardware viruses that can be spread from machine to machine. Some of the attacks are also very simple. Since flash memory can only be rewritten a certain number of times, a virus that simply rewrites it many times can render it worthless.

It has been some time since I have written commercial code, most of what I have written recently has been for training purposes. After reading this book, I have begun a crash program of writing code that demonstrates security flaws and have used it in my courses. If I ever go back to managing a coding team, no one will write a line of code before we cover this book in the finest possible detail. Without question it will be on my list of the best books of the year 2004.