Hiding Your Wares: Transparently Retrofitting Memory Confidentiality into Legacy Applications

Jamie Levy Bilal Khan
Department of Mathematics and Computer Science
John Jay College of Criminal Justice
New York, NY
Email: {jlevy,bkhan}@jjay.cuny.edu

***Abstract*— Memory scanning is a common technique used by malicious programs to read and modify the memory of other programs. Guarding programs against such exploits requires memory encryption, which is presently achievable either by (i) re-writing software to make it encrypt sensitive memory contents, or (ii) employing hardware-based solutions. These approaches are complicated, costly, and present their own vulnerabilities. In this paper, we describe new secure software technology that enables users to transparently add memory encryption to their existing software, without requiring users to invest in costly encryption hardware or requiring programmers to undertake complicated software redesign/redeployment. The Memory Encryption and Transparent Aegis Library (METAL) functions as a shim library, allowing legacy applications to transparently enjoy an assurance of memory confidentiality and integrity. The proposed solution is tunable in terms of trade-offs between security and computational overhead. We describe the design of the library and evaluate its benefits and performance trade-offs.**

I. Introduction

Many people doing secure programming in UNIX or UNIX-like environments are painfully surprised by the existence of `/dev/mem` and `/dev/kmem`. Together, these device files permit the root user to access arbitrary contents of physical memory and kernel memory, respectively—byte addresses in `/dev/mem` are interpreted as physical memory addresses, while byte addresses in `/dev/kmem` are interpreted as kernel virtual memory addresses. There is nothing one can do to prevent access to these device files, since from a kernel perspective, root is omnipresent and omniscient. Unfortunately, the ability to read and write to arbitrary memory makes it quite feasible for malicious programs to violate the implicit memory confidentiality and integrity assumptions made by other processes.

The problem of memory confidentiality has received considerable recent attention in the context of the study of "data lifetimes" [1]. Data lifetime researchers have noted that an application's sensitive data is often scattered widely through user and kernel memory, and continues to reside there for indefinite periods of time [2] , even *long after the program terminates*. In contrast, our research here considers the problem of application data confidentiality *during the lifetime of the application*; the data lifetime problem is solved as a corollary.

Certain well-established secure software design principles [6] attempt to address application data confidentiality issues. For example, it is common knowledge [7] that sensitive data such as cryptographic keys and passwords should be zeroed in memory immediately after they are known to be no longer needed. Unfortunately, these judicious strategies are too frequently disregarded by application, web browser and web server programmers. As a consequence, most consumer software faces increased risk given that sensitive user data is exposed in memory once a system has been compromised—this data can be easily examined by reading the previously mentioned memory device files, or inducing memory core dumps [8]–[11] by triggering program bugs. This paper describes a mechanism by which memory confidentiality can be provided transparently to existing legacy applications by the user themselves, without mandating programmers to redesign or even recompile their applications.

Reading/writing application heap memory is the foundation of many nefarious exploits, as illustrated by the prolific HOW-TO literature published within the hacker community (see e.g. [4], [14]). One illustrative case of this is Joseph Corey's paper [4] which targets the Honeynet project's Sebek intrusion detection system [12]. Since Sebek is intended to function as an IDS, its effectiveness hinges on remaining hidden from the would-be attacker. Corey illustrates that Sebek can be detected by searching for particular "landmark" patterns in memory, and illustrates how by overwriting memory at specific relative offsets from these patterns, program variables can be altered in a manner that disables IDS functions. Corey's attack is immune to Address Space Layout Randomization (ASLR)–a computer security feature which involves arranging the positions of key data areas, usually including the base of the executable and position of libraries, heap, and stack, randomly in a process' address space. By scanning memory contents to obtain relative address information, Corey's exploit sidesteps the ASLR features supported by many operating systems such as OpenBSD, Adamantix, Hardened Gentoo, Linux (via PaX, Exec Shield, etc.), Windows (via Wehnus, BufferShield, etc.) In this paper we describe a system which dynamically encrypts heap memory, making it nearly impossible for the attacker to find landmark patterns in memory, and hence preventing determination of which location(s) in memory to overwrite.

The prototype system is called METAL, the Memory Encryption and Transparent Aegis Library. METAL is a shim

[3] Jim Chow, Ben Pfaff, Tal Garfinkel, Mendel Rosenblum. Shredding Your Garbage: Reducing Data Lifetime Through Secure Deallocation, 14th USENIX Security Symposium (Security 2005).
[4] Corey, Joseph Advanced Honey Pot Identification and Exploitation http://www.phrack.org/fakes/p63/p63-0x09.txt
[5] Cowan, Crispin et al. PointGuardTM: Protecting Pointers From Buffer Overflow Vulnerabilities http://www.ece.cmu.edu/ adrian/630-f04/readings/cowan-pointguard.pdf
[6] J. Viega. Protecting sensitive data in memory. http://www-106.ibm.com/developerworks/security/library/ s-data.html
[7] J. Viega and G. McGraw. Building Secure Software. Addison Wesley, 2002.
[8] Coredump hole in imapd and ipop3d in slackware 3.4. http://www.insecure.org/sploits/slackware.ipop.imap.core.html.
[9] Security Dynamics FTP server core problem. http://www.insecure.org/sploits/solaris.secdynamics.core.html.
[10] Solaris (and others) ftpd core dump bug. http://www.insecure.org/sploits/ftpd.pasv.html.
[11] Wu-ftpd core dump vulnerability. http://www.insecure.org/sploits/ftp.coredump2.html
[12] The Honeynet Project. Know your Enemy. http://www.honeynet.org/papers/sebek.pdf
[13] Information and Communication Theory Group What is Electric Fence? http://genlab.tudelft.nl/old/html/helpdesk/software/efence/
[14] Dark Overload, Unix Cracking Tips, Phrack Volume Three, Issue 25, File 5 of 11, March 17, 1989.