ABSTRACT

Malware causes damage by stealing confidential data or making other software unusable. Ensuring software trustworthiness is difficult because malware may disguise itself to appear benign or trusted. This chapter explores the problem of making software more trustworthy through the use of binary integrity mechanisms. The authors review the problem of devising an effective binary integrity protection, and discuss how it complements other operating system security measures. They analyze design factors for binary integrity and compare existing systems. The authors then present a prototype which exemplifies a mandatory binary integrity mechanism and its integration within an operating system. Their system, BinAuth, demonstrates a practical, lightweight in-kernel binary authentication system for Microsoft Windows. A system like BinAuth shows that mandatory authentication is practical on complex
INTRODUCTION

Malware is a critical security threat today. A report by F-Secure (F-Secure, 2007) indicates that the amount of malware grew by 100% during 2007, and that there was as much malware produced in 2007 as in the previous 20 years altogether. A recent report from Organisation for Economic Co-operation and Development (OECD) (OECD, 2008) highlights a worrying trend that malware has now evolved from occasional exploits to a global multi-million dollar criminal industry, and is threatening the Internet economy.

Many of the system security attacks stem from the fact that distrusted code is executed on the system. A modern operating system has numerous built-in security measures designed to ensure a secure execution environment. The security measures are aimed at preventing illegal operations on a system, including illegitimate addition or modification of executable code on the file system. Due to various software vulnerabilities, such as buffer overflow (Cowan, Wagle, Calton, Beattie & Walpole, 2000) or format string vulnerability (Scut, 2001), a program however can be susceptible to local or remote attack. An attacker who succeeds in hijacking the execution of a process can perform subsequent operations under the context of the victim process. If the victim process happens to be in an elevated privilege when the attack occurs, for example root privilege due to SUID root feature in Unix/Linux, then the attacker gains unrestricted root access on the system.

Once an attacker succeeds in compromising a system, the next step is usually to install/modify executable code on the victim’s file system. This could be done for several reasons: install a back-door, plant spyware, or as a step for subsequent privilege-escalation attacks. A mechanism to deal with illegal addition or modification of executables on a system is thus a useful additional line of defense. Such a mechanism can also help prevent social engineering attacks which attempt to trick a user to install a software package that illegitimately replaces important system libraries. In more stringent environments, the protection mechanism can operate on controlled host systems to limit users to only run an approved set of executable files.

This chapter reviews the problem of establishing trust on the integrity of binary executables prior to their execution. We view file integrity mechanism as an important component for achieving a high level of software trustworthiness given the prevalence of malware and system attacks. In the rest of the chapter, we refer to any executable code stored in the file system as a binary. The goal of a software integrity protection system is to ensure that an executed binary only comes from trusted software providers/vendors, and that it is executed in the correct context. Later, we show that this can be efficiently achieved on complex operating systems such as Microsoft Windows.

A binary authentication system can provide the following authentication guarantees:

(i) **Binary-content authentication**: only binaries with previously known and trusted contents are allowed for execution; and
(ii) **Binary-location authentication**: a binary’s pathname must match its purported content.