Delayed and Controlled Failures in Tamper-Resistant Software

Gang Tan[1], Yuqun Chen[2], and Mariusz H. Jakubowski[2]

[1] Computer Science Department, Boston College
gtan@cs.bc.edu
[2] Microsoft Corporation
{yuqunc,mariuszj}@microsoft.com

Abstract. Tamper-resistant software (TRS) consists of two functional components: tamper detection and tamper response. Although both are equally critical to the effectiveness of a TRS system, past research has focused primarily on the former, while giving little thought to the latter. Not surprisingly, many successful breaks of commercial TRS systems found their first breaches at the relatively naïve tamper-response modules. In this paper, we describe a novel tamper-response system that evades hacker detection by introducing delayed, probabilistic failures in a program. This is accomplished by corrupting the program's internal state at well-chosen locations. Our tamper-response system smoothly blends in with the program and leaves no noticeable traces behind, making it very difficult for a hacker to detect its existence. The paper also presents empirical results to demonstrate the efficacy of our system.

1 Introduction

Software tampering continues to be a major threat to software vendors and consumers: Billions of dollars are lost every year to piracy[1]; tampered software, appearing legitimate to untrained consumers, also threatens their financial security and privacy. As the main countermeasure, the software industry has invested heavily in Tamper-Resistant Software (TRS) with varying degree of success. This paper focuses on a neglected aspect of tamper resistance, namely how the TRS should respond to tampering.

Software tampering is often conducted on a malicious host that is under a hacker's complete control: the hacker is free to monitor the hardware, as well as modify and observe the system software (i.e., OS). On current PC platform, without dedicated hardware support such as provided by NGSCB [6,17], TRS must rely on software obfuscation to evade detection and defeat hacking attempts [8,9,10,11,19]. Stealth, or the art of hiding code in the host program, is

[1] According to studies [1] by Business Software Alliance (BSA) and International Data Corporation (IDC), the retail value of pirated software globally is 29 billions, 33 billions, and 34 billions, in 2003, 2004, and 2005, respectively.

J. Camenisch et al. (Eds.): IH 2006, LNCS 4437, pp. 216–231, 2007.

the first and the *primary* defense that most TRS systems deploy against hackers. Ideally, the code pertaining to tamper resistance should be seamlessly intertwined with the host program's code, so that a hacker cannot discover its location(s) by either inspecting the program's code or monitoring its runtime behavior [7].

A TRS system consists of two functional components: *tamper detection* and *tamper response*; each can be made of multiple distinct modules. Both components are equally important to the effectiveness of a TRS system. In practice, however, most R&D work has gone into hiding the tamper-detection code, which verifies the host program's integrity [5,7,13]; surprisingly little has been done to improve the stealth of the tamper-response component. Since hackers tend to look for the weakest link to crack the defense perimeter of a TRS system, inadequate tamper-response mechanisms have often become the Achilles' heel of commercial TRS systems [4].

While some TRS systems can be effective if properly applied, software authors have often used only simple or default TRS features. For example, certain dongle- and CD-based copy protections perform just one or a few boolean checks, which may be easily patched out [14]. Thus, it is highly useful to automate the process of separating checks from responses.

In this paper, we describe a novel tamper-response system that evades hacker detection by introducing delayed, probabilistic failures in a program. The main technique is to corrupt certain parts of the host program's internal state at well-chosen locations so that the program either fails or exhibits degraded performance. One can also plug other failure-inducing techniques into our framework; some of them can be found in Section 6. Our tamper-response system smoothly blends in with the program and leaves no noticeable traces behind, making it very difficult for a hacker to detect its existence.

The rest of this paper is organized as follows. We describe some prior art and related work in Section 2. In Section 3, we introduce principles for effective tamper-resistant software. We describe our tamper-response system in Section 4. Implementation details and system evaluation are presented in Section 5. We discuss interesting extensions in Section 6, and conclude in Section 7.

2 Related Work

As informal advice, the idea of separating tamper detection from response has long been familiar to programmers of software-protection schemes [4]. The concept of "graceful degradation", or slow decay of a program's functionality after tamper detection, is a closely related technique, which has been widely reported to be used commercially [16]. Software authors typically have not revealed how specific implementations achieve these effects; in general, manual and application-specific techniques have been used. Our work provides systematic, automated methods of separating detection from response in general programs.

Commercial copy protection, licensing, and DRM systems have employed many unpublished techniques, which have been described by hackers on a large