Computers and Mathematics with Applications 65 (2013) 1326–1338

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Cherub: Fine-grained application protection with on-demand virtualization☆

Hai Jin [a,b,c,d,*], Ge Cheng [e], Deqing Zou [a,b,c,d], Xinwen Zhang [f]

[a] Services Computing Technology and System Lab, China
[b] Cluster and Grid Computing Lab, China
[c] School of Computer Science and Technology, China
[d] Huazhong University of Science and Technology, Wuhan, 430074, Hubei, China
[e] Xiangtan University, Xiangtan, Hunan, China
[f] Huawei Research Center, Santa Clara, CA, USA

ARTICLE INFO

Keywords:
Lightweight virtualization
VMM
On-demand protection

ABSTRACT

Cherub is an on-demand virtualization mechanism aiming to provide fine-grained application protection in untrusted environments. By leveraging late launch technology, Cherub dynamically inserts a lightweight *virtual machine monitor* (VMM) under a commodity *operating system* (OS) when critical pieces of an application code or data are to be processed. The novel design of Cherub with a double-shadowed page table extends VMM level memory protection into application level, such that it can isolate selected memory pages of a target process from the rest and other processes in the same OS environment. With this, Cherub enables fine-grained memory access control and therefore flexible security objectives. Compared to existing approaches, Cherub has the benefits of small code size, low performance overhead, no change to existing applications and commodity OS, and selective protection capability within a single application space. We implement Cherub in Linux and our analysis and evaluation demonstrate its effectiveness and practicality.

1. Introduction

A commodity *operating system* (OS) usually includes not only a kernel but also device drivers and many system services, which consist of the *trusted computing base* (TCB) for applications. Rich device drivers and system services generally comprise a large body of code with exploitable bugs or misconfigurations. As they run with the same privilege as the kernel, by exploiting their vulnerabilities, an attacker can gain complete access to system resources and compromise any application's security objectives. In addition, applications running in a commodity OS are weakly isolated and it is likely that they can affect each other. It has been widely recognized that current security mechanisms in commodity OS do not provide adequate protection for applications [1–4].

Several mechanisms have been proposed and developed to ameliorate the above problem. TCG-based attestations aim to verify an OS's integrity status [5,6] by measuring the entire OS components at boot-time and application codes at load-time, and securely storing and reporting to an attesting party. However, these mechanisms usually have very coarse-grained

☆ This paper is supported by National Base Research Program of China (2007CB310900), National Science Foundation of China (60973038), Wuhan City Programs for Science and Technology Development (201010621211), Innovation Research Foundation of Huazhong University of Science and Technology (2011TS074), and Key Lab of Information Network Security, Ministry of Public Security.

* Corresponding author.
E-mail address: hjin@hust.edu.cn (H. Jin).

0898-1221/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2012.02.001

measurement and verification, and they cannot detect any compromising during runtime of the system. Another set of work uses a secure coprocessor [7] or an extended processor architecture [8] to secure system runtime and data storage. Since these solutions demand major changes of platform architecture and resource management, they are not so viable for commodity OS.

Virtualization offers strong isolation between several domains or *virtual machines* (VMs) on a single physical platform, and therefore provides secure runtime environment for applications which handle sensitive transactions [9–14]. However, as virtualization is originally motivated and designed for consolidating servers and planning resources, these approaches usually introduce heavy performance overhead caused by virtual machine monitors (VMMs or hypervisors), which manage resource isolation and schedule tasks between VMs. Therefore, even if a user does not need high security environment for every application or for all the execution time a sensitive application, he has to bear the cost and performance overhead of virtualization. Furthermore, the concern of the trustworthiness for general purpose VMMs arises as they have grown both in code size and functional features, which make them like a traditional OS with large TCB [10,11,15]. Consequently, similar to commodity OS, VMMs are prone to design and implement vulnerabilities, and therefore their isolation and security functions might be compromised by attacks from guest OS [16–18].

Recent approaches have been proposed to enhance application security by isolating the trust of applications from the underlying OS components with advanced functions of hypervisors [19,15,20,21]. However, these approaches have several issues towards transparent and fine-grained application protection (cf. Section 7). For example, Overshadow does not provide the capability of protecting selective pieces of an existing application [20], which is desired for many complex applications where different trust should be given to more-critical and less-critical pieces of a single application. Furthermore, some of these approaches [15,20] encrypt complete application memory pages, which bring performance penalty as the size of a single application can be from dozens of KBs to many MBs. Flicker and TrustVisor provide fine-grained protection of critical code and data of an application [19,21]. However, they change the application programming model as the target code directly runs on nearly bare hardware. In addition, executing multiple security sessions in an application necessitates the suspension of a guest OS many times and has to rely on security storage offered by the *trusted platform module* (TPM) [22] to transfer parameters between them, which introduces significant performance overhead.

To address these problems, we propose Cherub, a transparent and fine-grained application protection mechanism in commodity OS without significant performance sacrifice to end users. Different from traditional virtualization-based protections, Cherub inserts a *lightweight virtual machine monitor* (LVMM) under a native OS during runtime in an *on-demand* manner, with the latest late launch and hardware virtualization techniques. The LVMM maintains a *protected execution environment* (PXE) to isolate the memory of a target application piece (code or data) from the rest of the application and other processes in the same OS environment. By controlling accesses to code and data in PXE, Cherub can enforce flexible security objectives.

Cherub achieves the capability of fine-grained memory access control through a technique called *double-shadow page table*, which leverages the memory virtualization function from the LVMM. From a high level view, Cherub provides multiple views of a guest OS's memory and checks the entry/exit points to a PXE. The protected code and data of a target application are defined by a set of access permissions which are associated with a set of cryptographic keys. Cherub measures, encrypts, and decrypts memory pages according to pre-defined access permissions. Only encrypted memory contents are exposed to external programs including operating system components.

Compared with existing approaches, Cherub has several unique properties: (1) Cherub has a dynamic chain of trust, and, to the best of our knowledge, it is the first VMM implementation built on late launch technology; (2) The virtualization layer of Cherub can be created and removed dynamically during the operating system runtime, and therefore the system does not need to bear the overhead of virtualization when it does not run any high-security applications; (3) Cherub has very small code size so it introduces much smaller TCB than general purpose VMMs and very small performance overhead during launching and runtime stages; (4) Cherub does not require modification to OS and the current programming model, and therefore it can protect a wide range of unmodified legacy applications running on an unmodified commodity OS, which cannot be supported by para-virtualization and co-processor based approaches; (5) Cherub provides flexible protection of an entire application or critical sections of an application via fine-grained memory access control.

The next section gives a brief description of hardware virtualization and late launch technology, and also presents the threat model of Cherub and our design goals, and Section 3 presents the design details. In Section 4, we illustrate the implementation of Cherub using Intel's TXT technology and its protection capability. Section 5 presents the evaluation results of our implementation on an unmodified Linux kernel. We present related work in Section 6 and conclude this paper in Section 7.

2. Background

2.1. Hardware virtualization

While software virtualization techniques have been maturing and widely deployed [23,9], hardware vendors are rapidly embracing virtualization and developing new features to simplify virtualization support. Intel's Virtualization Technology (VT-x) [24] and AMD's AMD-V [25] both adapt privileged instructions with a new CPU execution mode to allow a VMM to run in a new root mode below ring 0. A control transfer into the VMM is called a `VMExit` and control transfer to a VM is