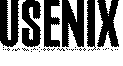
USENIX协会

委员会的议事程序

第13届USENIX安全研讨会

圣地亚哥，加利福尼亚州，美国

2004年8月9日至13日



©2004由USENIX协会版权所有。有关USENIX协会的更多信息，请访问：

电话：1 510 528 8649传真：1 510 548 5738电子邮件：office@usenix.org网址：http://www.usenix.org

个人论文的权利属于作者或作者的雇主。

允许出于教育或研究目的对作品进行非商业性复制。

此版权声明必须包含在复制文件中。USENIX承认此处的所有商标。

**基于TCG的系统设计与实现**

**完整性度量体系结构**

*雷纳·赛勒、张晓兰、特伦特·杰格和利安德特·范多恩*

*IBM T.J.沃森研究中心*

*纽约州霍桑市天际大道19号，邮编：10532*

{}*赛勒、张欣、杰格特、李安德特@沃森ibm.com*

**摘要**

本文介绍了一个安全的Linux完整性度量系统的设计与实现。加载到Linux系统上的所有可执行内容在执行之前都会进行度量，并且这些度量由Trusted Platform Module（TPM）保护，TPM是Trusted Computing Group（TCG）标准的一部分。我们的系统是第一个将TCG信任度量概念扩展到从BIOS到应用层的动态可执行内容的系统。实际上，我们表明，许多Microsoft NGSCB保证可以在今天的硬件和软件上获得，并且这些保证不需要新的CPU模式或操作系统，而仅仅依赖于独立的受信任实体（例如TPM）的可用性。我们将我们的信任度量体系结构应用于一个web服务器应用程序，在该应用程序中，我们展示了我们的系统如何检测不需要的调用，例如rootkit程序，并且我们的度量体系结构在所采取的度量的数量和进行度量对性能的影响方面是实用的。

# 1        介绍

随着自主计算、网格计算和按需计算的引入，越来越需要能够安全地识别在远程系统上运行的软件堆栈。对于自主计算，您需要确定已在服务器上安装了正确的修补程序

给定系统。对于网格计算，您关心的是广告中的服务是否真的存在，以及系统是否受到危害。对于按需计算，您可能会担心您的外包合作伙伴正在提供服务级别协议中规定的软件设施和性能。还有一种情况是，您正在与您的家庭银行或图书销售webservices应用程序进行交互，并希望确保它没有被篡改。

以上场景的问题是，你相信谁会给你这个答案？它不能是程序本身，因为它可能会被修改，给你错误的答案。出于同样的原因，我们不能信任运行这些程序的内核或BIOS，因为它们也可能被篡改。相反，我们需要回到一个不可变的根来提供答案。这本质上就是安全引导问题[1]，尽管在我们的场景中，我们感兴趣的是软件堆栈的完整性声明，而不是确保数字签名的遵从性。

可信计算组（TCG）定义了一组标准[2]，描述了如何对系统进行完整性度量，并将结果存储在一个独立的可信协处理器（可信平台模块）中，该协处理器的状态不会受到潜在恶意主机系统的危害。这种机制称为可信引导。与安全引导不同，此系统只进行测量，并由远程方来确定系统的可信度。其工作方式是，当系统通电时，它将控制权转移到一个不变的基础上。这个基础将通过对BIOS的下一部分内容计算SHA1安全散列来度量BIOS的下一部分，并使用TPM来保护结果。然后将此过程递归地应用于下一部分代码，直到操作系统已引导。

TCG可信引导过程由一组有序的顺序步骤组成，并且只定义到引导加载程序。从概念上讲，我们希望维护到应用程序层的信任度量链，但与引导过程不同，操作系统处理各种各样的可执行内容（内核、内核模块、二进制文件）。共享库、脚本、插件等）以及加载内容的顺序似乎是随机的。此外，操作系统几乎连续地加载可执行内容，并且在每次加载时测量内容会产生相当大的性能开销。

我们在本文中描述的系统解决了这些问题。我们已经修改了Linux内核和运行时系统，以便在可执行内容加载到系统中之后，但在执行之前，立即进行完整性度量。我们在内核中保存了一个有序的度量列表。我们稍微改变了TPM的角色，并使用它来保护内核中列表的完整性，而不是直接保存度量。为了向远程方证明加载了什么软件堆栈，系统需要使用TCG认证机制和这个有序列表来呈现TPM状态。然后，远程方可以确定有序列表是否被篡改，并且一旦列表被验证，（只是通过TPM 来证明列表没有被篡改，而不能证明度量值的可信性，系统运行软件的可信性，如果软件一开始就是恶意软件呢？如何辨别？）就可以确定它与度量关联的信任类型。为了最小化性能开销，我们缓存测量结果，并且只要不改变可执行内容，就可以消除将来的测量计算。我们对Linux系统所做的修改很少，大约4000行代码。

我们的增强功能跟踪系统执行的所有软件组件。独特组件的数量惊人地少，系统很快进入稳定状态。例如，作者使用的运行Redhat9的工作站，其工作负载包括撰写论文、编译程序和浏览web，其累积的测量条目不超过500个。在一个典型的web服务器上，累计测量值约为250。因此，对正在运行的软件堆栈进行完全指纹识别的概念非常容易处理。

**贡献：**本文的贡献如下：

•使用标准（商品）硬件的非侵入式可验证远程软件堆栈认证机制。

•动态可执行内容的有效测量系统。

•易处理的软件堆栈证明机制，不需要新的CPU模式或新的操作系统。

**大纲：**接下来，我们将介绍一个典型的运行时系统的结构，在本文中我们将建立一个完整性度量体系结构。在第三节中，我们介绍了诚信保护体系和认证领域的相关工作。在第4节和第5节中，我们将描述我们的方法的设计及其在标准Linux操作环境中的实现。第6节描述了一些实验，这些实验强调了在验证度量列表时，我们的解决方案是如何使完整性破坏可见的。它还总结了运行时开销。最后，第7节概述了我们正在实施或计划的体系结构的增强。我们的结果显示并验证了我们的架构是有效的，可以根据元素的数量进行扩展，成功地识别完整性破坏，并为扩展和未来的实验提供了一个有价值的平台。

# 2        问题陈述

为了提供完整性验证服务，我们首先检查系统完整性的含义。然后，我们描述一个web服务器示例系统，以确定必须解决的问题类型，从而以高度的可信度证明远程系统的完整性。我们表明，操作系统缺乏上下文，以提供必要的完整性测量的水平，但具有硬件的信任根，操作系统可以是完整性度量的基础。目前，我们推测发现完整性缺陷比完全验证更合适，但我们的目标是定义一个最终可以扩展以满足我们的度量需求的体系结构。（？？？所以问题是什么）

## 2.1        完整性背景

我们的目标是使远程系统（挑战者）能够证明另一个系统（证明者拥有的证明系统）上的程序具有足够的完整性。程序的完整性是一个二进制属性，它指示是否以未经授权的方式修改了程序和/或其环境。这种未经授权的修改可能导致程序的不正确或恶意行为，因此挑战者依赖它是不明智的。

虽然完整性是一个二进制属性，但完整性是一个相对属性，它取决于验证者对程序保护自身能力的看法。Biba定义了当程序依赖于（即，读取或执行）低完整性数据时，完整性受到损害[3]。在实践中，程序经常处理低完整性的数据而不被破坏（但不是所有的程序，一直如此），所以这个定义太有限了。ClarkWilson定义了一个模型，其中完整性验证过程在系统启动时验证完整性，高完整性数据仅由转换过程修改，转换过程经认证可保持完整性，即使其输入包含低完整性数据[4]。不幸的是，申请的认证费用太高，不实用。

最近的工作主要集中在度量代码和将完整性语义与代码相关联上。IBM4758明确定义了程序的完整性由程序及其祖先的代码决定[5]。在实践中，这种假设是可行的，因为程序及其配置是以可信的方式安装的，它与使用可由其他程序修改的文件相隔离，并且假设它能够处理来自外部系统的低完整性请求。为了使这个保证可信，ibm4758环境被限制为一个具有良好定义的输入状态的单一程序，并且完整性通过安全引导来实现。然而，即使这些假设也不足以防止4758上运行的应用程序受损，因为4758不能正确处理低完整性输入[6]。因此，进一步测量低完整性输入及其影响似乎是可能的。

本文的主要区别在于：（1）我们致力于在一个潜在的不可信方的控制下，为一个灵活的传统系统环境定义实际的完整性；（2）我们利用的唯一特殊硬件是可信计算组的可信平台模块（TCG/TPM）提供的信任根。在第一种情况下，我们可能不会简单地通过检查散列来假设所有程序都已正确加载，因为不受信任的一方可能会尝试更改程序使用的输入数据。例如，许多程序允许在命令行中指定配置文件。最终，应用程序定义了它们所使用的输入的语义，因此，如果应用程序的环境由不受信任的一方控制，则操作系统很难检测到应用程序是否以适当的方式使用了所有输入。但是，仅操作系统就可以发现许多漏洞，操作系统收集和保护度量值是非常重要的。

第二，ibm4758的专用硬件环境支持安全引导和内存锁定，但是这些功能要么不可用，要么不适用于当前的PC系统。安全引导并不实用，因为完整性要求不是固定的，而是由远程挑战者定义的。如果远程方能够确定系统的安全引导属性，那么系统将容易受到严重的拒绝服务威胁。相反，TCG/TPM支持可信引导（trusted boot），在可信引导中，验证系统被测量，并且这些测量被挑战者用来验证他们的完整性需求。由于受信任的引导不会在加载低完整性进程时终止引导，因此在“不受信任”的引导过程中，所有数据都可能受到攻击。由于多个应用程序可以同时在自主访问控制环境中运行，因此很难确定系统（例如数据库）的动态数据是否仍然可以接受。自由选择的完整性机制，如密封存储[7]，一般不能解决这个问题。

## 2.2        例子

我们以运行apachewebserver和tomcatweb容器的服务器为例，这些容器提供静态和动态内容，以便向运行在远程系统上的客户机销售书籍。系统运行的是RedHat 9.0 Linux环境。图1说明了影响Web服务器的运行时环境。

系统通过引导操作系统启动。引导过程由BIOS、grub引导加载程序和内核配置文件（/boot）决定/grub.conf公司). 前两者可以任意改变系统，因此必须对它们进行测量。有趣的一点是，配置文件的度量，例如grub.conf公司，是不必要的，只要它们不：（1）修改已加载的代码，以及（2）度量基础结构可以看到所有后续文件加载。由于BIOS和grub引导加载程序不受影响，我们只需要确保内核和其他程序的负载是由配置触发的。

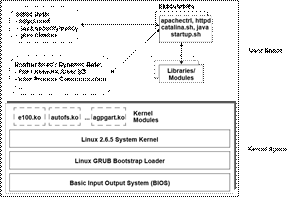


图1：运行时系统组件

引导过程会导致运行特定的内核。有各种不同类型的内核、内核版本和内核配置，它们决定了实际引导的系统。例如，我们从/boot/vmlinuz-2.6.5-tcg加载linux2.6.5tcg，其中包括一个TPM驱动程序和我们的度量挂钩。此外，内核可以由可加载的内核模块扩展。度量基础设施必须能够度量内核和加载的任何模块。挑战者必须能够确定这个特定的内核是否启动，动态加载的模块是否满足所需的完整性要求。

一旦启动内核，就可以运行用户级服务和应用程序。在Linux中，程序的执行从加载一个合适的解释器开始（例如，动态加载程序，如老苏)基于可执行文件的格式。目标可执行文件的代码和支持库的加载由动态加载程序完成。在我们的实验系统上，可执行文件包括以下文件：

•Apache服务器（apachectl、httpd等）

•Apache模块（mod访问.so，型号授权，型号cgi.so公司, ...)

•Tomcat servlet机器(启动.sh, 卡塔琳娜.sh，java，…）

•动态库(libjvm.so文件, libcore.so文件, libjava.so文件，libc2.3.2.so，libssl.so文件.4, ...)

所有这些代码都会影响系统完整性，因此我们需要对它们进行度量。内核知道何时加载可执行代码，因为相关文件是使用可执行标志进行内存映射的。但是，当内核模块从文件系统加载时，内核无法识别它们，因为它们是由modprobe或insmod等应用程序加载的，并且只有在将它们加载到内存后才被映射为可执行文件。最后，内核不知道何时将可执行脚本加载到bash之类的解释器中，因为它们是作为普通文件读取的。

应用程序本身加载的一些其他文件也定义了其执行行为。例如，定义servlet和web服务的Java类文件必须进行度量，因为Tomcat服务器加载它们来创建动态内容，例如购物车或支付页面。应用程序配置文件，如Apache的启动文件(httpd.conf文件)Tomcat（启动脚本）也可能改变Web服务器的行为。我们示例系统中的这些文件包括：

•Apache配置文件(httpd.conf文件)

•Java虚拟机安全配置

(java.security文件, java.policy策略)

•servlet和web服务库(axis.jar文件, servlet.jar，wsdl4j.jar，…）

虽然这些文件中的每一个都可能具有挑战者可以识别的标准内容，但很难确定应用程序实际使用了哪些文件以及用于什么目的。即使http.conf文件具有预期的内容，可能无法按预期加载。例如，Apache有一个命令行选项来加载不同的文件，文件系统中的链接可能会导致加载不同的文件，并且在测量文件和加载文件之间可能存在竞争。因此，像Tripwire这样对密钥系统文件的[8]测量是不够的，因为证明系统的用户（证明者）可能会更改实际确定其完整性的文件，并且这些用户不一定受到质疑者的信任。与动态加载程序一样，打开文件的完整性影响只有请求程序知道。然而，与动态加载程序的情况不同，确定应用程序加载的完整性影响的问题涉及更多程序的插装，并且这些程序可能具有不同的信任级别。

Web服务器环境的完整性还取决于运行可执行文件所使用的动态非结构化数据。关键的问题是，即使应用程序知道这些数据会影响其完整性，其度量也是无用的，因为挑战者无法预测将保持完整性的值。在web服务器示例中，关键的动态数据是：（1）来自远程客户端、管理员和其他servlet的各种请求；（2）图书订单数据库。需要确定的是订单数据或管理员命令是否只能由高完整性程序（即Biba）修改，以及低完整性请求是否可以转换为高完整性数据或被拒绝（即Clark Wilson）。密封存储不足以确保第一个属性，通常需要基于强制策略的信息流，而强制执行第二个属性需要受信任的升级程序或对应用程序本身的信任。

## 2.3        测量系统

基于对web服务器示例的分析，我们列出了实现ClarkWilson级完整性验证所必须完成的任务类型。

•：除非流程之间的信息流受到强制性限制，否则必须衡量所有流程的完整性。否则，影响流程的完整性范围可能会缩小到只有那些流程依赖于高完整性代码和数据的流程。**验证范围**

•：对于每个进程，无论是由操作系统、动态加载程序还是应用程序加载，执行的所有代码都必须具有足够的完整性。**可执行内容**

•：对于每个进程，其内容具有可识别的完整性语义的数据可以按照与上述可执行内容相同的方式处理。但是，我们必须确保捕获操作系统、动态加载程序和应用程序实际加载的数据。**结构化数据**

•：对于每个进程，其内容没有可识别的完整性语义的数据，数据的完整性取决于修改数据的进程的完整性，或者可以通过显式升级进程或此进程（如果符合克拉克-威尔逊意义上的转换过程）来升级数据的完整性。**非结构化数据**

第一条语句指出，对于使用自主策略的系统（例如NGSCB），必须度量所有进程的完整性，因为所有进程都可能相互影响。其次，我们必须度量所有代码，包括模块、库和应用程序以特殊方式加载的代码，以验证单个进程的完整性。第三，一些数据可能具有类似于代码的完整性语义，因此可以这样处理。第四，动态数据不能作为代码进行验证，因此需要数据历史记录、安全策略等来确定其完整性。挑战者可能会假设某些代码可以处理低完整性数据作为输入。对特定代码处理低完整性数据的能力缺乏正确的理解是当前许多安全问题的根源，因此我们最终希望清楚地确定如何使用低完整性数据。（如何来判断被修改的数据是恶意篡改还是人为修改？？？）

此外，我们架构的一个重要部分是挑战者的能力，以确保度量列表：

•新的和完整的，即包括截至认证执行时间点的所有测量，

•不变，即指纹确实来自加载的可执行和静态数据文件，没有被篡改。

已损坏的证明程序可以尝试通过更截断度量或传递改的度量来隐藏已损坏其状态的程序来进行欺骗。重放旧的度量列表相当于隐藏新的度量。

分析表明，柔性系统环境下的完整性验证是一个需要多个协调任务的难题。一种更实用的方法不是一次解决所有问题，而是提供一种可扩展的方法，该方法现在可以识别一些完整性缺陷，并为将来构建合理的完整性验证奠定基础。这种方法的动机是静态分析研究人员在最近的工作中采用的方法[9]。这些工具不是用来证明程序的完整性，而是用来发现bug，并且可以扩展到将来发现其他更复杂的bug。发现完整性缺陷对于识别需要修补的代码、非法信息流或在没有适当保护措施的情况下使用低完整性数据也很有用。例如，挑战者可以验证证明系统是否正在为其当前应用程序使用高完整性代码。

在本文中，我们定义了用于度量代码和结构化数据完整性的操作系统支持。操作系统确保加载到每个单独的用户级进程中的代码得到度量，这是应用程序度量其他代码和数据的基础，可以为这些代码和数据定义完整性语义。因此，我们的体系结构确保系统的广度是可测量的（即所有用户级进程），但测量的深度（即随后将哪些内容加载到进程中）是不完整的，但它是可扩展的，因此可以进行进一步的测量以增加对完整性的信心。目前，我们没有度量强制访问控制策略，但是体系结构支持扩展以包含此类度量，我们正在研究如何有效地使用它们。

# 3        相关工作

相关工作包括以前为测量系统以改进其完整性和/或实现远程完整性验证所做的努力。前期工作的关键问题是：（1）安全引导和认证引导的区别；（2）已有完整性度量方法的语义价值。

安全引导使系统能够测量其自身的完整性，并在某个操作损害了该完整性时终止引导过程。Arbaugh[1]的AEGIS系统为在PC系统上实现安全引导提供了一个实用的体系结构。它使用签名哈希值来标识和验证引导过程中的每一层。如果无法验证哈希值，它将中止引导系统。安全引导不能使具有挑战性的一方验证引导进程的完整性（即，经过身份验证的引导），因为它只是测量和检查引导进程，但不会生成进程完整性的证明。

IBM4758安全协处理器[10]实现了安全引导和认证引导，尽管是在受限环境中。它通过在激活（flash）分区之前验证分区，以及在将可执行文件加载到系统之前强制执行有效的签名来保证安全的引导。一种称为outgoing authentication[5]的机制支持将每个后续层链接到其前一层的证明。前置层通过生成包含加密哈希和后续层的公钥的签名消息来证明后续层。为了保护一个应用程序不受其他应用程序缺陷的影响，一次只能运行一个应用程序。因此，应用程序的完整性取决于代码的散列和应用程序安装数据的手动验证。此数据只能在安装后由受信任的代码访问。我们的web服务器示例运行在一个更加动态的环境中，在这个环境中，多个进程可以访问相同的数据并进行交互。此外，质询方和认证方的安全要求可能不同，使得基于质询方的要求的安全引导是不切实际的。

Trusted Computing Group[11]是一个公司联盟，他们共同开发了一个用于Trusted Platform Module的开放接口，这是提供加密功能和受保护存储的系统的硬件扩展。默认情况下，TPM通过收集目标代码上的散列序列，支持在内容和顺序方面验证静态平台配置。例如，研究人员研究了如何使用TPM来证明一个系统已经启动了一个有效的操作系统[12]。在操作系统上运行的应用程序的完整性超出了本文的工作范围，这正是我们希望扩展TPM应用程序的地方。

Marchesini等人[13]描述了一种使用签名可信配置来保护系统完整性的方法。这种配置存储敏感配置文件的签名。每次打开真实文件时，所谓的强制执行器都会根据真实文件检查配置中签名文件的完整性。该方法通过对长期存在的服务器证书进行TPM密封以及将解封绑定到正确的配置来强制实现完整性。在这方面，工作与[12]中描述的平台配置有关。所有已知的现有工作都没有将软件堆栈的度量从静态引导配置无缝地扩展到应用程序级别。

Terra[14]和微软的下一代安全计算基础（NGSCB[7]）基于相同的硬件安全体系结构（TCG/TPM），并且在为认证引导提供“整个系统解决方案”方面是相似的。NGSCB将平台划分为可信和不可信两部分，每个部分运行自己的操作系统。只对可信部分进行了度量，这限制了方法的灵活性（并非所有感兴趣的程序都应完全可信），并且它依赖于尚未可用的硬件和基础软件。

Terra是一个可信计算体系结构，它围绕一个可信的虚拟机监视器构建，该监视器可以对虚拟机中运行的软件进行身份验证。Terra试图解决构建可信任的定制封闭盒运行时环境（例如ibm4758）和开放系统之间的冲突，开放系统提供丰富的功能和显著的规模经济，但是由于其灵活性而难以信任。因此，Terra试图以一种完全不同的方式来解决和我们一样的问题。Terra在分区块级别度量受信任的虚拟机监视器。因此，一方面，Terra在证明示例性4gb VM分区时产生约20mb的测量值（即散列）。另一方面，由于这些测量值代表块，因此很难解释不同的测量值。因此，我们的系统有选择地度量那些对动态运行时系统有贡献的系统部分；它在一个语义丰富的高级别上进行度量，并使远程方能够在文件级别上解释不同的度量。

# 4           完整性测量体系结构的设计

我们的完整性度量体系结构由三个主要组件组成：

•认证系统上的度量机制决定了运行时环境的哪些部分要度量、何时度量以及如何安全地维护度量。

•完整性挑战机制，允许授权挑战者检索计算平台的度量列表，并验证其新鲜度和完整性。

•完整性验证机制，验证度量列表是否完整、无篡改和新鲜，以及验证运行时组件的所有单个度量条目是否描述了可信的代码或配置文件。

图2显示了这些机制如何交互以启用远程认证。测量是由所谓的测量代理发起的，这些代理对文件进行测量，（a）将测量存储在内核中的有序列表中，（b）向TPM报告测量列表的扩展。

完整性质询机制允许远程质询者请求测量列表以及测量列表的集合（图2中的步骤1）。接收到这样的质询，证明系统首先从TPM检索签名的聚合（图2中的步骤2和3），然后从内核检索度量列表（图2中的步骤4）。然后，在步骤5中，两个都返回给证明方。最后，认证方可以验证信息

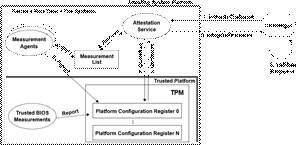


图2：基于TPM的完整性度量

第六步分析了认证系统运行时完整性可信的原因。

## 4.1        假设

在我们描述体系结构的这三个组件之前，我们先建立关于攻击者模型的假设，因为如果没有这样的限制，总会有攻击者能够欺骗远程客户端。

我们使用TCG标准[11]提供的服务和保护是为了：（1）使质疑方能够对认证系统（测量环境）的平台配置建立信任；（2）确保质疑方的测量环境编制的测量列表没有被篡改。我们假设TPM硬件按照TPM规范[11]工作，并且TPM正确嵌入到平台中，从而确保BIOS、引导加载程序和以下系统环境部分的正确测量。

TPM无法防止针对系统的直接硬件攻击，因此我们假设这些攻击不属于威胁模型的一部分。

我们假设代码度量足以描述其行为。因此，可以对自更改代码进行评估，因为代码更改自身的预期能力反映在度量中，并且可以在验证中加以考虑。对于被认为只有通过加载和卸载模块才能更改的内核代码也是如此。基于恶意DMA传输覆盖内核代码的内核更改不会被处理；但是，设置DMA的代码是经过测量的，因此需要进行评估。

我们还假设质询方持有有效且可信的证书，该证书绑定了认证系统TPM的公共RSA标识密钥。在使用这些寄存器验证测量列表之前，质疑方将使用这些寄存器验证认证系统TPM的引用寄存器内容。*AIKpub公司AIKpub公司*

我们假设对度量数据没有保密要求，而这些要求不能通过控制对认证服务的访问来满足。

最后，对于系统完整性度量的解释，我们依赖challenger的运行时，因为验证结果必须安全地计算、解释和执行。我们假设挑战者可以安全地决定信任哪些度量，方法是将它们与可信度量的列表进行比较，或者将决定卸载给根据列表签名可信度量的可信方

共同政策（即共同评价标准）。

## 4.2        测量机制

我们的度量机制包括加载新可执行文件时的基本度量，以及度量其他可执行内容和敏感数据文件的能力。其思想是BIOS和bootloader测量初始内核代码，然后使内核能够测量对自身的更改（例如，模块加载）和用户级进程的创建。对于用户级进程，内核使用相同的方法，它测量加载到进程中的可执行代码（例如，通过mmap加载的动态加载程序和httpd）。然后，该代码可以度量它加载的后续安全敏感输入（例如，由httpd度量的配置文件或脚本）。challenger的信任依赖于它对度量代码的信任，以度量它的安全敏感输入，保护自己不受未度量输入的影响，并在重新启动时保护它所依赖的数据。操作系统可以通过强制访问控制策略进一步保护应用程序，强制访问控制策略可以限制恶意、未测量输入的来源，并保护数据不受修改。然而，这种政策的使用是未来的工作。

在本节中，我们将讨论如何进行测量。第5节描述了这些测量在完整测量系统中的应用。

为了唯一地标识任何特定的可执行内容，我们在文件的完整内容上计算一个SHA1哈希。生成的160位哈希值明确地标识了文件的内容。不同的文件类型、版本和扩展名可以通过其独特的指纹加以区分。

各个散列被收集到一个度量列表中，该列表表示认证系统的完整性历史。不允许修改度量列表，因为这样会使攻击者隐藏与完整性相关的操作。由于我们的体系结构是非侵入式的，因此它既不能防止系统被破坏，也不能防止度量列表在事后被篡改。然而，为了防止此类恶意行为不被注意（防止损坏的系统作弊），我们在证明系统上使用了一个硬件扩展，称为Trusted Platform Module，以使对度量列表的修改对具有挑战性的方可见。

TPM[11]提供了一些受保护的数据寄存器，称为平台配置寄存器，只能通过两个功能进行更改：第一个功能是重新启动平台，清除所有PCR（值0）。第二个函数是函数，它将一个160位的数字和一个PCR寄存器的数字作为参数，然后通过计算一个SHA1（PCR[]| | n）来聚合和当前PCR[]的内容。这个新值存储在PCR[]中。基于我们的假设，即TPM硬件的行为符合TCG规范，并且没有发生直接的物理攻击，系统没有其他方法来更改任何PCR寄存器的值。*TPM扩展不我不我我我*

我们使用平台配置寄存器来维护体系结构所进行的所有测量的完整性验证值。在被测组件可能影响和潜在损坏系统之前，所采取的任何测量也会聚合到TPM PCR（使用）中。因此，在直接（可执行）或间接（配置的静态数据文件）进行控制之前，记录任何测量的软件。例如，如果已进行测量，则所选PCR中的聚合包含1（…SHA1（SHA1（0 | | m1）| | m2）| | mi）。TPM的受保护存储可防止设备或系统软件进行修改。虽然损坏的系统可以使用其他选择的值对其进行扩展，但扩展的计算方式（SHA1的属性）防止恶意系统调整PCR中的聚合以表示指定的系统。一旦恶意组件获得控制权，就太晚了，无法向认证方隐藏此组件的存在和指纹。*TPM扩展我米*1*…英里沙*

因此，损坏的系统可以操纵度量列表，但这是通过重新计算列表的聚合并将其与安全存储在TPM中的聚合进行比较来检测的。

## 4.3        诚信质疑机制

完整性质询协议描述了质询方如何从认证系统安全地检索度量和验证信息。在检索认证信息时，协议必须防止以下主要威胁：

•重放攻击：恶意证明系统可以重放系统损坏前的证明信息（度量列表+TPM聚合）。

•篡改：恶意认证系统或中间攻击者可以在将度量列表和TPM聚合传输到质询方之前或之后对其进行篡改。

•伪装：恶意的认证系统或中间攻击者可以用另一个（未被破坏的）系统的度量列表和TPM聚合替换原始度量列表和TPM聚合。

我们假设这个机制是在一个安全的（例如。，

SSL认证和保护）连接，以保证真实性和保密性要求。图3描绘了质询方用于安全地验证认证系统的完整性声明的完整性质询协议。在第1步和第2步中，创建一个不可预测的160位随机变量，并在质询请求消息中将其发送到。在步骤3中，认证系统将受保护的RSA密钥加载到TPM中。这是用所谓的存储根密钥（SRK）加密的，这个密钥只有TPM知道。TPM规范[11]描述了如何在TPM内部安全地创建2048位AIK，以及如何由受信任方安全地认证相应的公钥。此受信任方证书将PCR的签名链接到特定系统中的特定TPM芯片。然后，来自TPM芯片的请求a现在用私钥对所选（或多个PCR）和最初提供的进行签名。要完成步骤3，将检索所有度量的有序列表（在本例中是从内核）。然后，在步骤4中使用质询-响应消息进行响应，包括签名的聚合和nonce-in，以及声明的完整度量列表。*C级作为C级暂时的克雷克作为艾克艾克AIKpub公司作为引用聚合酶链反应暂时的C级艾克普里夫作为作为克里斯引用毫升*

1. *C级*:创建不可预测的160位*暂时的*2. *C级*→作为：克雷克(*暂时的*)

3a.：负载保护进入TPM*作为艾克普里夫*

3b.：检索={PCR，nonce}AIK*作为引用信号私人*

3c.：检索测量列表4。→C:ChRes（，）*作为毫升作为引用毫升*

5a.：确定受信任（AIKpub）*C级证书*

5b.：验证{PCR，nonce}AIK*C级信号私人*

5c.：验证和使用*C级暂时的毫升聚合酶链反应*

图3：完整性挑战协议

在步骤5a中，首先检索可信证书（AIKpub）。此AIK证书将的验证密钥绑定到特定系统，并声明仅此TPM知道相关的密钥，并且从未在未受保护的情况下导出。因此，通过将的唯一标识（这个唯一标识在哪给出？？？）与（AIKpub）中给出的系统标识进行比较，挑战方可以发现伪装。必须验证此证书是否有效，例如，通过检查受信任颁发方的证书吊销列表。然后在步骤5b中验证签名。*C级证书AIKpub公司引用作为证书C级*

在步骤5c中，验证的新鲜度，因此新鲜度的（测量聚合）。如果nonce匹配，只要第2步中的唯一且不可预测，就可以保证新鲜度。一旦两次接收到nonce或者可以预测nonce（或者甚至预测一个足够小的nonce将落入其中的集合），它就可以决定重放旧的度量或者使用预测的nonce提前请求TPM签名的报价。在这两种情况下，引用的完整性度量可能并不反映实际的系统状态，而是过去的状态。如果nonce提供的安全性不足，那么签名密钥的有效性可能会受到限制，因为签名聚合的重播窗口也绑定到使用有效的签名密钥。*C级引用聚合酶链反应暂时的作为毫升*

在步骤5b中验证签名可以检测到对TPM聚合的篡改，（？？？）因为它将使签名无效（假设今天数字2048bit签名的加密属性，假设只有TPM知道密钥，并且假设没有对TPM的硬件篡改）。在步骤5c中，通过遍历度量列表并重新计算TPM聚合（如第4.2节所述模拟TPM扩展操作），并将结果与步骤4中接收到的签名中包含的TPM聚合进行比较，可以看到篡改度量列表。如果计算的聚合与有符号的聚合匹配，则度量列表有效且无阻尼，否则无效。*C级毫升聚合酶链反应引用*

## 4.4        完整性验证机制

质疑方必须验证验证方平台配置的单独测量值，以及自重新启动后在验证系统上进行的动态测量值。配置和度量列表的集合已经在完整性质询协议中进行了验证，这里假设。TPM聚合的有效性也是如此。

判断是否信任一个认证系统是基于独立地测试每个度量列表条目，将其度量值与一个可信度量值列表进行比较。更复杂的验证模型可以将多个测量关联起来，以获得评估结果。无论条目是代码还是数据，测试度量条目在逻辑上都是相同的。其思想是，条目匹配一些具有已知完整性语义的预定义值。新程序版本、未知程序或以其他方式操作的代码可能会产生未知指纹。因此，具有挑战性的一方可以测量程序更新的指纹，并将其添加到数据库中；反过来，具有已知漏洞的旧程序版本[15]可能会被重新分类为不可信。

提出质疑的一方必须制定政策，说明如何对指纹进行分类以及如何处理未知或不可信的指纹。通常，如果没有额外的策略执行机制来保证隔离不信任的可执行文件，那么不信任的指纹会导致对整个认证系统的完整性的不信任。或者，可信的指纹可以由可信的第三方签名，例如，关于它们是否适合实施与其目的相关的某些安全目标（共同标准评估）。

**交易完整性**通常，当认证系统处理对具有挑战性的一方来说很重要的交易时，它的完整性很重要。为了验证正在质询方和认证方之间发生的事务的完整性（例如，Web请求），质询方可以在处理事务之前和之后，例如，在发送Web请求之前和在接收Web响应之后，质询认证系统的完整性。然后，通过将用于验证TPM报价的证书和用于在SSL连接设置期间作为Web请求的一部分对服务器进行身份验证的证书安全地链接，可以将认证和事务绑定到同一个系统。（？？？没明白）如果证明系统两次都是可信的，那么-看起来-交易也可以是可信的。

然而，这并不是完全正确的，因为它假设两个度量都发生在同一个时期（有效期），即整个事务中的任何系统更改都将记录在第二个度量中。然而，证明系统可能在第一次质疑之后和交易发生之前就遭到破坏。然后，证明系统就可以在第二次挑战发生之前重新启动。因此，尽管在两个时间点上都是可信的，但是重新启动覆盖了针对挑战者的不可信的证明系统状态。即使可能性看起来很小，系统也可以很快重新启动，并实际进入一个完全预定义的状态（因此显示出与早期测量相同的测量列表）[1]。

幸运的是，有一种方法可以发现一个时代是否发生了变化，即系统是否在两个证明之间重新启动。为此，我们可以使用所谓的TPM计数器。与PCR相反，这些计数器从不清除或减少，但只能在TPM的整个生命周期内增加。每次系统重新启动时，BIOS都会触发其中一个计数器的增加。BIOS还负责在计数器达到最大值时立即禁用TPM。典型的TPM有多个可以组合的计数器，因此足以满足正常的平台生命周期[2]。因此，在度量列表中包含这样一个计数器的可信内核可以确保在系统重新启动后，两个度量列表的前缀至少在这个单个计数器度量中不同。

因此，在此增强版本中，可以通过确保在事务之前的第一个质询处验证的度量列表是在事务之后的第二个质询处验证的度量列表的前缀来验证事务完整性。然后，系统没有重新启动，因此（根据我们的假设）任何可能影响认证系统上事务的不信任系统组件都将显示在第二个挑战的度量列表中。实际上，只要是非侵入性的，我们的体系结构就不能提供可预测的安全性，但是它可以提供系统完整性状态的追溯性保证。（？？？如何保证）

# 5        实施

本节描述了我们为实现度量功能而对Linux系统所做的增强。在启动任何动态测量之前（即，在linuxrc或init启动之前），我们的内核预先加载其测量列表（？？？），其中包含BIOS、引导加载程序、内核和initrd（如果适用）的预期测量值，并使用在预定义的TPM PCR中找到的真实引导进程的集合（？？？难道是TPM 来完成），作为我们自己衡量总量的起点。如果实际的引导过程与预期的不同，那么度量列表的验证将失败。我们关注在初始操作系统引导之后测量动态运行时内容的阶段。

我们的原型实现是在Redhat9.0Linux发行版上完成的，作为2.6.5内核的Linux安全模块（LSM）[3]。原型实现分为四个主要部分：将测量点插入系统以测量文件或内存（第5.1节）、测量文件或内存（第5.2节）、防止绕过测量（第5.3节），以及验证度量，以确保我们的体系结构的实现在认证系统上实际到位（第5.4节）。

## 5.1        插入测量点

在4.2节中，我们概述了度量方法，包括内核中的度量和用户级程序的度量。这里我们描述实现。

我们实现了基于Linux内核LSM接口的内核测量。使用file\_mmap LSM hook，我们在将可执行文件映射到虚拟内存之前对任何文件进行测量。

使用sysfs文件系统，我们允许用户空间应用程序通过将请求写入/sys/security/measure来发出度量请求，包括要度量的文件的文件描述符。使用内核加载模块例程，在重新定位加载模块之前，我们在加载模块的内存区域上引入一个度量调用。

在第4.2节中，我们概述了度量方法，其中度量的可执行代码本身（例如，shell）可以对其行为所依赖的加载文件内容（例如，shell命令文件）进行额外的度量。如果可执行代码的完整性不高，则会检测到它（因为它已经在度量列表中）。如果它具有高度的完整性，则可以信任它来测量其加载的数据（难道是要在程序中设置Hoop函数？？？）。

我们将在下面描述如何度量动态运行时负载，以及如何在使用过程中保护度量文件。

**用户级可执行文件：**用户级可执行文件通过用户级加载程序加载。当通过系统调用execve调用二进制可执行文件时，内核调用二进制处理程序例程，然后该例程解释二进制文件并为可执行文件定位适当的加载程序。然后内核将加载程序映射到内存中，并设置环境，以便当execve调用返回时，加载程序继续执行。加载程序依次执行进一步的加载操作，最后将控制权传递给目标可执行文件的主函数。在静态链接的二进制文件的情况下，唯一加载的文件是目标二进制文件本身，我们在映射前由内核调用的文件xmmap LSM hook中测量它。

**可动态加载的库：**动态链接的二进制文件通常需要加载它所依赖的其他库。这个过程由用户级加载程序完成，对内核是透明的。但是，链接器通过使用mmap系统调用将共享库（标记的可执行文件）映射到虚拟内存中，mmap系统调用总是调用文件xmmap LSM hook。因此，文件xmmap LSM hook instrumentation提供的中介产生所有静态和动态链接的可执行文件（包括共享库）的度量。

**内核模块：**内核模块是内核的扩展，可以在系统启动后动态加载。模块加载可以是显式的（通过insmod或modprobe），如果启用了自动模块加载，则可以是隐式的。在后一种情况下，当内核检测到需要某个模块时，它会通过在用户进程的上下文中调用modprobe自动查找并加载相应的模块。对于2.6内核，两个程序都将内核模块加载到内存中，然后调用sys\u init\u module系统调用来通知内核有关新模块的信息，然后将新模块复制到内核内存并重新定位。因此，内核模块可以在从文件系统加载时由insmod或modprobe在用户级别进行测量，也可以在它们驻留在内核内存中并重新定位之前在内核中进行测量。我们实现了两个版本。但是，我们更喜欢后一个版本，因为它可以防止利用内核加载程序insmod或modprobe中的漏洞（可能未知）篡改内核级代码的度量。因为没有合适的LSM钩子可用，所以我们在load\u module例程中添加了一个measure调用，该例程由init\u module系统调用来重新定位内存中的模块。

**脚本：**脚本解释器作为二进制可执行文件加载和度量。但是，解释器加载决定其行为的额外代码，因此我们更希望脚本解释器也能够测量与完整性相关的输入。目前，我们已经对bashshell进行了检测，以便在加载和解释任何解释过的脚本和配置文件之前对它们进行测量。这将所有服务启动脚本都包含在度量列表中。在引导Redhat9.0Linux并运行Gnome桌面系统的实验中，我们观察到大约60-70个bash脚本和源文件的度量。插入其他程序（Perl、Java）很简单，但我们预计需要应用程序程序员提供更多支持。

## 5.2        测量

本节描述在测量点使用的内核级测量调用的实现，以启动文件或内存区域的测量（对于内核模块）。measure调用采用一个参数，即指向包含要度量的文件的文件结构的指针。从文件结构中可以查找相应的inode和数据块，并在数据块上使用SHA1。

发出度量调用的地方有三个：（1）对用户级应用程序使用的伪文件系统/sys/security/measure的写入/存储例程的实现，（2）正在内存映射为可执行代码的文件mmap security LSM hook度量文件，（3）在内核模块代码被重新定位之前，加载模块例程。filemmap钩子接收文件指针作为参数，sysfs项的write例程接收文件描述符，使用fget例程从中检索文件指针。在properties参数中没有设置位的地方，我们忽略文件映射调用，因为这些文件不是映射的可执行文件。*保护执行器*

文件测量值与实际加载内容之间的一致性取决于：（1）加载的inode的准确标识和（2）对inode所描述的文件的任何后续写入的检测。对于内存映射的可执行文件，这两种情况都由内核处理。内核在测量时持有的保护锁确保文件只要映射为可执行文件，就不能被其他人写入。此锁在测量时由映射函数保持。模块是在它们已经在内核内存中时进行度量的，因此它们不容易受到这种不一致的影响。对于从用户空间测量的文件，我们假设测量应用程序将文件描述符（用于启动测量）保持打开状态，直到读取完内容或在重新打开文件时发出新的测量调用为止。这样可以确保测量的文件是实际读取的文件。第二，度量和读取用户级调用与另一个修改数据的写入调用之间可能存在竞争。我们将这种情况称为测量时间-使用时间（ToM-ToU）竞争条件，并在第5.3节中描述我们如何处理这种情况。但是，除非文件的完整内容在本地系统上得到缓存和保护，否则无法可靠地度量远程NFS文件。我们目前还没有实现这种缓存。

一个简单的度量实现是为每个度量调用获取一个指纹。但是，对于经常加载的可执行文件和库来说，这种方法会带来很大的性能开销（参见第6.2节）。

相反，我们使用缓存来减少性能开销。其思想是保留已执行的测量的缓存，并且仅当文件以前未被看到（缓存未命中）或者自上次测量以来文件可能已更改时才进行新的测量。对于后一种情况，我们只记录一个新的文件度量值，如果该文件实际发生了更改。每次应用程序运行时记录相同的度量值将严重影响列表的管理（存储、检索、验证）。内核模块总是在加载时在内存中进行度量，但是只有在度量列表中还没有度量时，才会添加它们的度量。

我们将所有测量值存储在一个单独链接的有序列表中。测量顺序对于检测测量列表的任何修改至关重要。如果未按顺序检查度量，则聚合哈希将与TPM扩展操作产生的TPM聚合不匹配。此外，我们还收集与所测量文件相关的元信息，例如文件名、用户ID、组ID或加载实体的安全标签，或文件系统类型，这些信息可能有助于评估加载此文件的影响或将其与本地安全策略相匹配。此时，我们的实现非正式地在度量列表中收集这些附加数据，但不将其包含在度量中。

为了提高效率，我们用两个哈希表覆盖链表，一个用被测文件的inode号和设备号键控，另一个用被测文件的结果指纹（SHA1值）键控。因此，可以通过遍历度量列表、通过其inode（仅用于文件度量）或通过其指纹来访问每个度量条目。measure调用使用与目标文件的文件描述符相对应的inode在哈希表中快速查找该文件，并查看它以前是否被测量过。

每个度量条目都包含一个脏标志位，指示文件是干净的（未修改）还是脏的（可能已修改）。我们将在下面描述度量的语义。

**测量新文件：**如果在inodekeyed散列表中找不到该文件，那么我们通过对其完整内容计算SHA1散列来度量该文件。此时，我们使用计算出的指纹来检查它是否存在于由现有度量的SHA1散列值键控的散列表中。如果没有找到测量的指纹，那么我们将创建一个新的测量条目，并将其添加到列表中并调整哈希表结构。最后，在从调用返回并允许加载可执行内容之前，我们通过SHA1散列扩展受保护的TPM硬件中的相关平台配置寄存器。如果之前已经测量过指纹，那么我们将从系统调用返回，而不扩展TPM或测量列表。如果复制可执行文件并因此产生相同的指纹，则可能发生这种情况。在本例中，我们假设这两个可执行文件是等价的。

**重新测量文件：**如果该文件是在inode键控哈希表中找到的，那么它是以前测量过的。如果找到的度量条目的dirty标志是CLEAN（CLEAN hit），则不需要执行任何操作，系统调用将返回。如果dirty标志位是dirty（dirty hit），那么我们计算文件的SHA1值。如果测量的指纹与存储在测量列表中的指纹相同，那么我们将重新设置脏标记。我们不扩展PCR或记录这种测量，因为这是已知的。

如果测量的指纹与inode的found测量条目中存储的指纹不同，那么我们使用SHA1值作为密钥在哈希表中查找新的指纹。如果SHA1值存在，则之前测量过相同的文件内容（当前文件的副本）。我们返回时不记录测量值，如上所述。如果哈希表中不存在SHA1值，则当前文件已更改。创建一个新的度量条目并将其添加到表中，并且在度量调用返回之前扩展PCR。

**脏标记：**每当目标文件（a）以写、创建、截断或附加权限打开时，（b）位于无法控制访问的文件系统（如NFS）上，或（c）属于已卸载的文件系统时，我们都将dirty标志位设置为dirty。这似乎有点保守，因为openforwrite（或卸载文件）不一定会导致对文件的修改。如果文件在具有写权限的打开后没有更改，SHA1键哈希表使我们能够清除dirty标志。如果我们控制对文件的访问，那么在这种情况下，我们会清除dirty标志。实验表明，在使用本地文件系统的非开发系统上，缓存上的脏命中率远远小于1%。

**测量核心模块：**每当内核模块准备集成到内核中时，我们都会发出一个度量调用。我们计算尚未重新定位的内核模块所在的内存区域的SHA1值，从而为每个内核模块生成一个独立于其最终内存位置的代表性度量。然后，我们在所有现有测量中使用SHA1键哈希表检查这个SHA1指纹是否已经在测量列表中。如果它是已知的，那么我们从度量调用返回。如果没有，那么我们从位于内存区域开头的ELF头中提取模块名，将度量作为新度量添加到度量列表中，最后扩展TPM寄存器以反映更新的度量列表。内核模块必须始终被度量，因为我们没有任何容易获得的信息来指示脏标志状态。但是，通常只加载了几个内核模块。或者，用户级应用程序insmod和modprobe可以在将内核模块加载到内存时测量文件。在这种情况下，它们的测量遵循前面描述的文件测量过程。

## 5.3        测量旁路保护

每当我们遇到我们的度量体系结构无法提供正确的度量或可能被绕过的情况时，我们都会通过使用随机值扩展TPM聚合而使其无效，而不扩展度量列表并删除随机值以保护它不被以后使用。因此，从此时起，对度量列表的聚合验证将失败。我们不干扰系统（非侵入性的），但是我们禁止这样一个系统在重新启动之前成功地进行认证。在我们的实验中，这些机制都不是在正常的系统使用过程中触发的，而只是由恶意或非常不寻常的行为触发的。

尽管我们假设没有针对TPM的硬件攻击，但我们设计了这样一个系统，即一个受到威胁的系统不能在未被发现的情况下更改度量列表，因为它不能成功地操纵TPM来覆盖软件中的此类攻击。因此，用TPM硬件支持我们的体系结构是有用的，而且是必要的，即使在（假设）没有物理攻击的情况下，为了发现欺骗系统。然而，任何拥有根标识的人都可以尝试通过不太为人所知的接口来改变系统，从而绕过我们的度量挂钩，从而破坏度量的有效性。因此，我们实现了一些故障保护机制来捕获这些工作并使TPM聚合失效（悲观地）。我们将在下面讨论其中的一些问题。

*测量时间使用时间竞赛条件：*从理论上讲，文件内容可以在测量时间和实际加载时间之间进行更改。Linux确实保护内存映射文件，但不保护通常加载的文件（例如脚本文件、配置文件）。因此，我们在一个测量文件的inode中实现了一个计数器度量计数，它跟踪指向这个inode的打开文件描述符的数量，在这个inode上引发了一个度量调用。我们在调用度量调用之前增加计数器（在/sys/security/measure节点的sysfs write实现中），在关闭被度量的文件描述符时减少计数器（使用file\u free\u security LSM钩子）。我们向inode\u permission LSM钩子中添加一个检查，该钩子捕获对相关inode的度量值计数为0的文件的写入或附加权限请求。在本例中，我们使TPM聚合无效，因为度量可能不反映实际加载的文件内容，但我们选择不干扰请求。我们认为任何这样的行为都是恶意的。*>*

*绕过用户级别测量。*为了确保应用程序发出的度量请求实际上会导致内核中的度量，我们必须确保

/sys/security/measure节点实际上是在写入时发布度量的节点。在度量列表中不留下可疑指纹的情况下绕过此问题的唯一方法是，首先防止系统挂载sysfs文件系统，或者在挂载后使用不可疑的程序（命令）将其卸载。我们通过确保在init启动之前（在内核启动时）装入sysfs来防止第一种情况，第二种情况是将sysfs保持在忙碌状态（锁定它），以便root用户无法卸载它。

*绕过脏标记。*以root身份运行的进程可能会尝试绕过脏标记，从而在测量和加载之间更改文件内容，或者尝试通过访问特殊的存储控制接口（例如/dev/hda）或内存接口/dev/kmem来更改（否则是不易受攻击的，因此是受信任的）内存中的应用程序或内核。我们捕获此类特殊情况并如上所述使TPM聚合无效。这是必要的，以防止内核在未测量此更改的情况下被更改。这种可疑案件在正常系统中很少有必要或观察到。

*正在卸载文件系统。*我们脏标记属于正在卸载的文件系统的任何度量，因为我们不再控制此文件系统上的更改。热插拔硬盘驱动器可以更改并重新插入更改的文件。为此，我们将文件的超级块指针保留在文件的度量结构中。在整个度量列表中遍历与装入点相关的脏标记条目会带来开销，但在大多数正确设置和配置的系统上，这种情况很少发生（例如，在关机时），度量列表通常不是很大（1000个条目）。*<<*

*测量函数之间的运行时错误。*如果在整个测量记录过程中出现任何错误，例如，由于分配新测量结构时内存不足错误或其他意外事件导致我们无法正确测量，我们将使TPM聚合失效。

总之，度量函数使用伪文件系统sysfs、内核LSM钩子文件mmap和load模块内核例程中插入的度量调用来为系统提供度量点。我们使用LSM hooks inode\_permission、sb\_umount、inode\_free\_security和file\_free\_security来实现脏标记并防止ToMToU竞争条件（通常是恶意的）。我们在文件中使用LSM安全子结构和inode内核结构来存储状态信息，例如脏标志和度量值计数。

## 5.4        验证测量值

我们的体系结构使用TPM的受保护存储来保护度量列表的完整性。一旦进行了度量，就不能更改或删除它，而不会导致度量列表的聚合哈希与TPM聚合不同。然而，质疑方还必须确保证明系统具有正确的度量体系结构，以便实际启动和执行所有必要的度量。由于我们的体系结构组件在执行时也会被度量，因此具有挑战性的各方可以通过检查这些度量来确定体系结构是否到位。

度量体系结构的主要部分在静态内核中。因此，具有挑战性的一方只信任实现度量体系结构核心部分的内核。其他内核对于具有挑战性的一方来说是不可接受的，因为它们可以跳过重要的度量。

如果插入指令的insmod和modprobe程序在将内核模块加载到内核之前对它们进行测量，那么只有插入了测量调用的内核模块加载程序才是可接受的。如果看到任何其他具有insmod功能的程序的指纹，则该程序必须不受信任，因此验证失败。这不适用于我们的情况，因为我们在内核中度量内核模块。如果我们要求shell程序在加载或执行脚本和源文件之前对它们进行度量，那么发现没有检测到度量调用的shell的指纹就不能被信任。根据第4.4节中描述的相应可执行文件的已知漏洞，可以信任系统任何其他部分的已知指纹。未知指纹可能是由于用户级程序的更改而导致的，这些程序被假定用于测量其输入（例如bash），或者是不可接受的输入文件，并且由于其相应程序的功能可能是恶意的，并且可能违反安全性假设，因此无法信任。

# 6        结果

## 6.1        实验

为了测试系统检测可能攻击的能力，我们使用流行的Linux rootkit lrk5构建了一个小实验。我们从一个完美的目标系统开始，对这个系统进行测量。然后，我们对目标系统发起rootkit攻击，并在攻击后再次进行测量。图4（a）显示了良好系统的（部分）度量列表，图4（b）显示了被rootkit破坏的同一系统的相应列表。斜体的条目显示，攻击后，syslogd程序的签名不同，表明rootkit已用特洛伊版本替换了原始syslogd。此示例说明了如何使用我们的系统可靠地发现此类攻击。

## 6.2        绩效评估

我们检查通过以下方式调用的度量调用的性能：（i）内核文件\u mmap LSM hook，（ii）内核加载\u模块函数，以及（iii）用户空间应用程序将度量请求写入/sys/security/measure。

#000:D6DC07881A7EFD58EB8E9184CCA723AF4212D3DB引导集合

#001:CD554B285123353BDA1794D9ABA48D69B2F74D73 linuxrc

#002:9F860256709F1CD35037563DCDF798054F878705纳什

#003:84ABD2960414CA4A448E0D2C9364B4E1725BDA4F初始

#004:194D956F288B36FB46E46A124E59D466DE7C73B6 ld-2.3.2.so#005:7DF33561E2A467A87CDD4BB8F68880517D3CAECB libc-2.3.2.so。。。

*#110:F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2系统*...

**（一）**

...

#110:F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2系统日志。。。

*#525:4CA3918834E48694187F5A4DAB4EECD540AA8EA2系统*...

**（二）**

图4：检测Rootkit攻击。

我们首先检查文件xmmap LSM安全钩子的开销，它度量所有可执行内容和动态库。到目前为止，这是最常被称为和最敏感的性能指标。为了确定文件\u mmap LSM度量钩子的延迟，我们从用户级别度量mmap系统调用的延迟，该级别调用这个文件\u mmap LSM钩子。我们的延迟测量（包括映射和取消映射）考虑了三种不同的情况，即no\_SHA1、SHA1和SHA1+extend。no \u SHA1表示文件\u mmap在缓存中发现目标是干净的情况。在很少观察到的SHA1情况下，重新测量目标文件并重新计算SHA1指纹。但是，不会扩展TPM，因为发现指纹已在缓存中。SHA1+extend表示当测量一个全新的文件时，结果指纹需要扩展到TPM芯片中。例如，这种情况在系统启动时或系统更新后更为常见。因为目标是测量延迟，所以我们使用2字节大小的测试文件。微观基准的实施基于HBench框架[16]。表1显示了结果。

|  |  |  |
| --- | --- | --- |
| mmap类型 | mmap延迟（stdev） | 文件mmap LSM |
| 不，沙一号 | 1.73 *µ*s（0.0） | 0.08 *µ*s码 |
|  | 4.21 *µ*s（0.0） | 2.56 *µ* |
|  | 5430 *µ*s（1.3） | 5430 *µ* |

表1：文件mmap LSM hook的延迟（文件大小2字节）。

出于参考目的，我们包含了mmap系统调用的运行时间，而不调用文件\u mmap LSM measurement hook。从表中可以清楚地看出，在干净缓存命中（no \u SHA1）的情况下，文件\u mmap LSM hook的开销是最小的—需要0.08（1.73-1.65）s才能运行。它只不过是从要映射的文件的inode读取脏标志信息。幸运的是，我们的经验表明，这是大多数情况下，即使是服务器往往运行很长时间，占所有措施调用的99.9%以上。*µ*

重新测量文件（SHA1）时，mmap系统调用大约需要4.21秒，相对于参考值大约需要2.5秒的开销。本例显示了设置测量文件和在哈希表中搜索匹配指纹的开销。注意，这种情况并不能度量指纹本身的开销，因为文件大小只有2个字节。指纹性能将在后面讨论。extend操作显然是最昂贵的，大约需要5毫秒来执行。这是可以理解的，因为extend操作与TPM芯片交互并创建一个新的度量列表条目。如前所述，这两种情况合计不到所有度量调用的0.1%。因此，我们有信心——而且我们的经验也证实了——我们的系统为测量可执行文件而对用户施加的性能损失可以忽略不计。*µµ*

从用户级调用度量包括（i）打开/sys/security/measure，（ii）编写度量请求，以及（iii）关闭

/系统/安全/措施。此方法适用于测量配置文件或解释的脚本文件（例如bash脚本或源文件）。与文件\u mmap LSM hook一样，我们在这里也区分了没有\u SHA1、SHA1和SHA1+extend的三种情况。结果见表2。这个

|  |  |  |
| --- | --- | --- |
| 通过sysfs进行测量 | | 开销（stdev） |
| 测量 |  |  |
| 参考 | 系统fs打开/写入/关闭 | 4.32 *µ*s（0.0） |

表2：通过sysfs的用户级测量的延迟（文件大小2字节）。

在无SHA1情况下，用户级测量延迟为4.32秒。这种开销主要是与文件系统相关的开销*µ*

–打开、写入、关闭/sys/security/measure，如表2中的参考值所示。no\u SHA1案例的度量相关开销在上下文切换和文件系统相关开销中消失。解释其他测量值很简单。

如第5.1节所述，可以通过两种方式测量内核模块：通过用户级应用程序insmod和modprobe，或者通过在init\u module系统调用的load\u module函数中重新定位内核模块之前引入一个测量例程。通过insmod或modprobe测量它们将内核模块测量性能转移到用户级测量领域，开销如表2所示。在load\u module内核函数中测量内核模块的延迟与在文件\u mmap LSM measurement hook中测量可执行内容的延迟几乎相同。但是，因为内核模块在重新定位之前已经在内存中了，所以没有脏的标记信息，我们没有干净的命中，只有SHA1或SHA1+extend的情况。我们考虑内核模块加载一个不频繁的、时间较短的事件，因此从安全的角度（见第5.1节）建议在内核中进行度量。

接下来，我们将指纹识别性能表示为文件大小的函数。我们在SHA1情况下测量mmap系统调用的运行时间，改变输入文件的大小。这包括表1所示的纯mmap系统调用的1.65秒的参考开销。结果见表3。当文件大小较大时，指纹开销可能很大。例如，测量一个128KB的文件大约需要1.5毫秒。随着文件大小的增加，运行时间几乎呈线性增长。这些延迟转化为大约每秒80MB的吞吐量性能。*µ*

|  |  |
| --- | --- |
| 文件大小（字节） | 开销（stdev） |
| 2 | 4.21 *µ*s（0.0） |
| 512 | 10.3 *µ*s（0.0） |
| 1公里 | 16.3 *µ*s（0.0） |
| 16千 | 197 *µ*s（0.1） |
| 12.8万 | 1550 *µ*s（1.1） |
| 1米 | 12700 *µ*s（16） |



表3:SHA1指纹操作的性能与文件大小的关系。

通过测量内存内核模块，我们预计在计算SHA1时的吞吐量要比在表1中描述的文件xmmap LSM hook中测量文件（首先必须从磁盘读入内存）稍微好一些。然而，我们的测量结果只比表3中所示的file\mmap情况稍微好一点。我们用Linux文件缓存效应来解释这一点。在同一个文件上使用热缓存进行多次测量，这使得测量开始时，几乎完整的文件已经驻留在文件缓存中。这也表明表1中的吞吐量数字对于文件度量来说应该是乐观的。

这些实验是在ibmnetvistam桌面工作站上用一个包含1000个条目的测量列表进行的，其中包括一个Intel奔腾2.4ghz处理器和1gbyte内存。停止的所有非必要服务。

## 6.3        实现和可用性方面

我们的内核实现包括用于测量、脏标记和旁路保护的LSM钩子，包括4755行代码（loc）和注释。这段代码驻留在自己的security/measure内核目录中，因此只要LSM接口不变，就很容易移植到新的Linux内核版本。我们需要在kernel/module.c的load\_module例程中添加另外2个loc来度量内核模块的加载。为了检测bashshell，我们在加载源文件或解释脚本文件的位置插入2个loc。这些用户级度量调用基于42 loc的头文件，该头文件将用户级度量请求宏转换为正确的/sys/security/measure写入。将架构从2.6.2移植到2.6.5linux内核大约需要10分钟。从2.4内核中的非LSM实现到2.6内核中基于LSM的完整性度量体系结构版本，大大降低了实现的复杂性并提高了可移植性。

我们已经成功地将完整性度量体系结构堆叠为SELinux之上的LSM模块，这需要对SELinux进行一些小的修改，以调用我们的钩子并共享文件和inode内核结构中的安全子结构。这些变化很小，但它们是必要的，因为当前的linuxlsm实现将大部分堆栈实现留给了模块本身。

我们的经验表明，一个标准的redhat9.0linux系统，包括Xwindow服务器和Gnome桌面系统，在运行大约一周后，累积了大约500-600个度量条目，包括大约60-100个bash脚本和源文件度量。这些bash度量涵盖了所有bash服务启动和关闭脚本以及本地源代码脚本（例如∧\.bashrc）。我们的度量体系结构引入的开销可以忽略不计，即使在系统引导时，大多数度量都被记录并扩展到TPM中。因此，我们相信我们的性能结果能够代表正常的Linux环境。

# 7        讨论

我们的架构是非侵入性的，不能防止系统运行恶意程序。然而，我们也修改了我们的方法来加强安全性。在本例中，我们使用一组可信程序的预期指纹预加载度量缓存。然后，测量调用对要测量的文件进行指纹识别，并将其与预期的指纹集进行比较。如果指纹与其中任何一个都不匹配，它将中止加载并报告非法指纹。请注意，证明系统的强制执行要求可能不同于质询者的强制执行要求，因此质询者仍然需要执行验证。

我们的度量体系结构并不局限于度量可执行代码。将度量挂钩添加到应用程序中，我们可以将结构化输入数据（如配置文件和java类）包含到我们的度量中。更改很简单—插装应用程序（如Apache或Java classloader）意味着在加载相关文件之前添加一个度量调用。

为了建立对系统的信心，我们的方法会影响隐私。认证协议发布认证系统的详细信息，允许挑战者或受信任的第三方建立信任。但是，认证系统对这些信息的发布有完全的控制权，并且可以运行它信任的代码来不发布这些信息。此外，系统代理可以配置为向经过身份验证的挑战者发布证明，并且操作系统只能向该代理提供报价。

在加载的可执行文件中引入频繁更改可能会导致度量列表超出实际限制，从而导致拒绝服务攻击。为了防止这种攻击，可以配置度量列表的最大长度。任何额外的测量值都会聚合到受TPM保护的PCR寄存器中，但测量值不会存储在内核中。因此，超过此最大测量次数的系统将无法成功说服具有挑战性的各方其完整性，因为测量列表将不再针对集合进行验证。

# 8        结论

介绍了一个安全的Linux完整性度量系统的设计与实现。该系统将TCG信任概念从BIOS一直扩展到通用操作系统的应用层。我们使用钩子扩展操作系统，以测量何时将第一个代码加载到进程中（file\u mmap LSM hook），提供一个measure sysfs条目以请求后续测量，并检测何时对测量的inode进行了更改。除了执行的文件之外，这个机制还支持动态加载程序、共享库和内核模块的度量。此外，该方法是可扩展的，因此应用程序可以测量其专用负载，如bash所示。结果是，我们表明，许多Microsoft NGSCB保证可以在今天的硬件和软件上获得，并且这些保证不需要新的CPU模式或操作系统，而仅仅依赖于独立可信实体的可用性。这样的系统已经可以检测到各种完整性问题，例如rootkit或易受攻击的软件的存在。我们的测量结果表明，非开发系统可以实际测量，并且测量开销是合理的。

该测量系统是可扩展的，我们相信，我们最终可以实现超出微软NGSCB的保证。强制访问控制策略的应用可以确保动态数据不能被修改，除非是受信任的来源[17]。识别低完整性数据流可以控制是否应该允许这些流，是否可以在系统级或应用程序内对其进行有效限制。

我们目前正在将完整性度量体系结构实现的源代码作为开源公开，并努力将其作为可选的LSM内核模块集成到内核中。

# 致谢

作者要感谢ibmlinux技术中心的持续和宝贵的支持，感谢我们来自ibmtoko研究实验室的同事，特别是seijimunetoh和他的同事，感谢他们有趣的讨论和grub引导加载程序的TPM增强。最后，我们要感谢罗纳德·佩雷斯、史蒂夫·巴德和匿名裁判的宝贵意见。

# 工具书类

[1] W.A.Arbaugh，D.J.Farber和J.M.Smith，“一个安全可靠的引导架构”，在IEEE计算机协会安全和隐私会议上发表。IEEE，1997年，第65-71页。

[2] 可信计算组http://www.trustedcomputinggroup.org。

[3] K.J.Biba，“安全计算机系统的完整性考虑”，技术代表MTR-3153，Mitre公司，Mitre公司，贝德福德，马萨诸塞州，1975年6月。

[4] D.D.Clark和D.R.Wilson，“商用和军用计算机安全策略的比较”，在IEEE安全和隐私研讨会上，1987年。

[5] S.W.Smith，“可编程安全协处理器的传出身份验证”，载于ESORICS，2002年，第72-89页。

[6] M.Bond，“对加密处理器事务集的攻击”，2001年加密硬件和嵌入式系统研讨会论文集，2001年5月。

[7] P.England、B.Lampson、J.Manferdelli、M.Peinado和B.Willman，“可信开放平台”，IEEE计算机，第36卷，第7期，第55-62页，2003年。

[8] G.Kim和E.Spafford，“Tripwire的经验：使用完整性检查器进行入侵检测”，《系统管理、网络和安全会议III》，USENIX，1994年。

[9] D.Engler、B.Chelf、A.Chou和S.Hallem，“使用特定于系统的程序编写的编译器扩展检查系统规则”，《第4届会议论文集》，2000年10月。*操作系统设计与实现研讨会（OSDI 2000）*

[10] J.Dyer、M.Lindemann、R.Perez、R.Sailer、L.van Doorn、S.W.Smith和S.Weingart，“构建IBM 4758安全协处理器”，IEEE计算机，第34卷，第10期，第57–66页，2001年。

[11] 可信计算组，可信平台模块主要规范，第1部分：设计原则，第2部分：TPM结构，第3部分：命令，2003年10月，版本1.2，版本62，http://www.trustedcomputinggroup.org。

[12] H.Maruyama、F.Seliger、N.Nagaratnam、T.Ebringer、S.Munetho和S.Yoshihama，《技术报告》，提交出版，2004年，提交。

[13] J.Marchesini、S.Smith、O.Wild和R.MacDonald，“试验TCPA/TCG硬件，或：我如何学会停止担忧和爱熊”，技术报告TR2003-476，达特茅斯PKI实验室达特茅斯学院，汉诺威，新罕布什尔州，美国，2003年12月。

[14] T.Garfinkel、B.Pfaff、J.Chow、M.Rosenblum和D.Boneh，“Terra:一个基于虚拟机的可信计算平台”，过程。第九届ACM操作系统原理研讨会，2003年，第193-206页。

[15] CERT协调中心，“CERT/CC咨询”http://www.cert.org/advisories。

[16] A.B.Brown和M.Seltzer，“Lmbench之后的操作系统基准测试：英特尔x86体系结构上NetBSD性能的案例研究”，1997年ACM SIGMETRICS计算机系统测量与建模会议论文集，1997年6月，第214-224页。

[17] T.Jaeger等人，“利用信息流进行完整性验证”，提交出版，2004年。

[[1]](" \l "_ftnref1" \o ")这在另一个TPM机制中使用，允许对平台配置保密，尽管最初这不包括任何动态测量。

[[2]](" \l "_ftnref2" \o ")TPM规范[11]要求外部可访问计数器必须允许每5秒递增7年，而不会导致硬件故障。

[[3]](" \l "_ftnref3" \o ")这里介绍的机制非常通用，因此移植到类Unix系统应该非常简单。