第 32 卷 第 22 期 计 算 机 工 程 2006 年 11 月
Vol.32 No.22 Computer Engineering November 2006
·安全技术· 文章编号：1000—3428(2006)22—0169—03 文献标识码：A 中图分类号：TP309

可信 Linux 关键组件验证方案的研究

叶 波，陈克非
（上海交通大学计算机科学与技术系，上海 200030）

摘 要：近几年可信计算方面的研究发展迅速，但在支持可信计算的 Linux 方面的研究却相对落后，无法对所有可能改变系统可信状态的关键组件进行完整性验证，以至于无法判断系统是否处于可信状态。为了弥补这个不足，该文提出了一种验证 Linux 关键组件的新方案。该方案基于 Demetrios Lambrou 的想法并对其进行了完善，弥补了其不能验证配置文件、动态共享库和可执行脚本的缺点，保证了对所有可能改变系统可信状态的关键组件的验证。

关键词：可信计算；信息安全；Linux

Research on How to Measure Trusted Computation Enabled Linux's Critical Objects

YE Bo, CHEN Kefei
(Department of Computer Science and Technology, Shanghai Jiaotong University, Shanghai 200030)

【Abstract】 These years saw rapid development of research on trusted computing. But research on trusted computing enabled Linux is not enough yet. Not every system critical object can be measured nowadays, which causes it is impossible to judge whether the system is trusted. To supply the gap, this paper represents a scheme to measure Linux's system critical objects. This scheme based on the idea of Demetrios Lambrou and improved it. The problem that configuration files, dynamic shared library and executable scripts can not be measured has been fixed in this scheme. Now all components that maybe change system's trusted state can be measured.

【Key words】 Trusted Computation; Information security; Linux

计算机安全问题的根本原因在于现有 PC 本身的不安全性。当初设计 PC 时就没有考虑安全性，缺乏很好的硬件防御措施，使得现有的安全方案都是纯软件的，缺乏硬件上的支持，从而十分脆弱。

为了解决这个问题，2000 年 12 月，由 Intel、Microsoft、HP 和 IBM 等公司组成了 TCPA(Trusted Computing Platform Alliance)、现已更名为 TCG(Trusted Computing Group)组织，提出了一种改进现有 PC，使其成为可信赖计算平台的方法。

现在，无论是TPM(Trusted Platform Module)芯片还是完整的可信计算机平台都已经有多个国内外厂商研制成功；在软件方面，Linux 2.6.12 内核已经包含了TPM芯片的驱动。同时IBM正在开发Linux下的TSS(TCG Software Stack)，为其他应用程序使用TPM提供统一的接口。Applied Data Security Group开发了trusted Grub[3]，它能够对自身的各个阶段及Linux内核进行验证，可以作为可信的OS loader。不过，到目前为止还没有看到对所有可能改变系统可信状态的关键组件的完整性验证方案，只有Demetrios Lambrou提出了一个初步的想法[2]。为了弥补这个不足，本文将提出一个完整的可信Linux关键组件的验证方案。

1 可信计算概念

可信计算平台的基本思路是基于可信任链的想法。

1.1 可信任链

可信任链是由一个可信任根(节点 0)出发，由这个可信任根来验证另一个节点 1，从而这个节点 1 也是可信任的了；再由这个可信节点 1 来验证另一个节点 2，从而节点 2 也是可信任的了；如此反复直至节点 N。使得可信任节点集合从最初的{节点 0}逐步增长为{节点 0，…，节点 N}。其中可信节点 i(i=0,1,…,N-1)验证节点 i+1 从而使原本不可信任节点 i+1 也成为可信任节点的过程称为信任传递。

1.2 可信计算平台结构

可信计算平台包含 3 个可信任根：可信任度量根(Root of Trust for Measurement, RTM)，可信任存储根(Root of Trust for Storage, RTS)，可信任报告根(Root of Trust for Reporting, RTR)。可信任度量根用于生成可靠的完整性度量，由物理模块 CRTM(Core Root of Trust for Measurement)实现。可信任存储根和可信任报告由物理模块 TPM 实现。如图 1 所示。

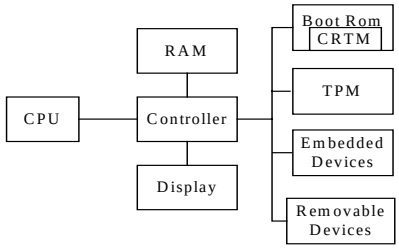

图 1 可信计算 PC 平台结构

从物理上来讲，可信计算平台与传统计算平台主要不同之处就是增加了 CRTM 和 TPM 模块。

CRTM是可信计算平台初始化代码中的不变部分。可信平台复位时必须从CRTM开始执行[1]。

作者简介：叶 波(1980－)，男，硕士、助教，主研方向：可信计算；陈克非，教授、博导

收稿日期：2006-02-21 **E-mail：**yebo9098@263.net

TPM 主要完成密码计算及数据、密钥的安全存储和使用，实现对环境度量信息的签名和报告，是数据存储和信息报告信任的初始点。

1.3 可信计算平台的启动过程

当 CRTM 就是整个 BIOS 时，首先，CRTM 验证(包括了度量与比较两个步骤)OS 装载器，如果可信则把执行权交给 OS 装载器；然后由 OS 装载器验证 OS，如果可信则把执行权交给 OS；最后由 OS 来验证应用程序是否可信，如果可信则开始执行。也就是说，在系统启动过程中，CRTM 将信任传递给 OS 装载器，OS 装载器将信任传递给 OS，OS 再把信任传递给应用程序。

在验证过程中，各程序的散列值会被存储在平台配置寄存器 (Platform Configuration Registers, PCR)中，而且在关机之前这部分 PCR 的值不能被重置，只能被扩展。当远端或本地程序(请求者)需要验证当前系统是否处于预定义的可信状态时，这些 PCR 的值就会被读取。请求者验证了可信计算芯片对这些 PCR 值的签名之后，与预先存储的值相比，就可以知道该系统运行的程序是否是预先规定的可信程序。

2 可信 Linux 关键组件验证方案

在平台刚启动时候 BIOS、OS 装载器、内核的运行是严格串行的。但是在 Linux 系统启动了之后情况却并非如此：在 Linux 系统中，可能改变系统可信状态的有内核及其模块、二进制可执行文件、共享库、配置文件、可执行脚本(perl,python,shell,…)等，它们的执行都是非严格串行的。这正是可信 Linux 研究中的一个难点。

对于 Linux 内核，在运行的时候很可能改变。在这样的情况下，即使 OS Loader 验证了一开始运行的那个内核是可信的，由于内核模块随时可以被插入或卸载，这种可信状态也随时可能被破坏，原先 PCR 中的度量值也不再代表当前运行的内核情况。这就需要当插入内核模块时验证其是否可信，当卸载内核模块时验证该操作是否会影响内核可信状态。

Linux 是多任务操作系统，多个应用程序可以同时运行，所以同样不能用前者验证后者然后把执行权交给后者的方法。为了保证系统的可信状态，需要在每个可执行文件装载前判断该文件及其执行参数是否可信。

同样，对于其他可能改变系统可信状态的对象也需要在装载前进行验证。

2.1 Demetrios Lambrou 提出的基于 LSM 的验证方法

LSM[4](Linux Security Modules)是一个轻量级的通用的访问控制架构。在内核很多地方设置了钩子函数，在打开文件、装载程序等操作之前需要检验权限。于是Demetrios Lambrou提出可以创建基于LSM的模块keeper，利用LSM的这些钩子函数来验证内核模块、二进制可执行文件[2]。

在这种方案中，需要维护一个可信列表。这个可信列表是文件路径及文件经签名散列值的二元组的集合，如

/bin/sh 467d705e6f540bbb8c5d70e57af15f5328504ccc

这个列表需要在一个被隔离的可信系统上获得，以确保这些经签名散列值所对应的组件的确是可信的。当系统启动的时候，这个列表会被装载到内核里面。

当试图装载一个内核模块时，Keeper 模块事先注册好的 LSM 钩子函数会被调用。这个函数对该内核模块取散列值并与从可信列表中计算得到的散列值进行比较，如果匹配就允许加载；如果不匹配的话，就可以根据预定义策略或拒绝加载，或虽然允许加载但同时对 PCR 的值进行扩展，使该系统处于不可信的状态。

在装载一个二进制可执行文件时同样对其取散列值及比较，并根据结果与策略做相应的动作。

2.2 Demetrios Lambrou 方案的不足之处

Demetrios Lambrou 方案虽然简单易实现，但适用范围也较狭窄，仅适用于内核模块和二进制可执行文件，并不适用于配置文件、动态共享库和可执行脚本。

动态共享库的装载是由 Glibc 完成的，LSM 中并没有合适的钩子函数可用。如在打开一个动态共享库的时候就对其进行验证，就会在很多不需要验证动态共享库时进行验证工作，由于验证失败可能导致拒绝打开文件，从而可能带来不必要的麻烦。其实动态共享库只有真正要执行的时候才需要被验证，比较可行的方法是在验证二进制可执行文件完后对该可执行文件将要用到的动态共享库进行验证。但是，在验证完动态共享库跟真正装载动态共享库并执行之间还有一个时间差，在这段时间内，动态共享库可能被恶意修改。所以需要将这个动态共享库标记为不可修改，只有当所有使用了这个动态共享库的程序都退出了，该动态共享库才可被修改。

对于配置文件，也没有合适的钩子函数可用，也不能在打开配置文件时就对其进行验证。只有当程序真正将要运行的时候，该程序对应的配置文件才需要被验证，应该在验证程序之后对其配置文件进行验证。这时也需将配置文件标记为不可修改，只有使用了该配置文件的程序退出了才能把标记改回可修改状态。

对于可执行脚本，情况更为复杂。比如一个 perl 脚本，它由 shell 调用 perl 解释器来对这个脚本解释执行。在 LSM 中同样没有合适的钩子函数可以利用。如果像动态共享库和配置文件一样在验证完 perl 解释器之后就对 perl 脚本进行验证，就需验证所有 perl 脚本，这样不仅性能大大降低，而且一个并不运行的不可信 perl 脚本会导致所有 perl 脚本被拒绝运行。本文的解决方法是由已经过验证的 shell 程序来验证可执行脚本。比如前面提到的 perl 脚本，shell 在调用 perl 解释器之前对该脚本进行验证，并将其标记为不可修改，然后再调用 perl 解释器来对该脚本解释执行。

2.3 新的改进方案

如图 2 所示，改进方案包括 Keeper2 模块和可信 shell 两部分代码以及 4 个列表。

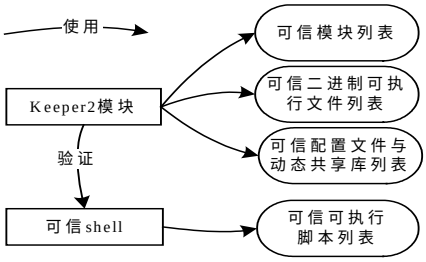

图 2 新方案的结构

可信模块列表的格式与 Demetrios Lambrou 方案中的可信列表一样，也是文件路径及文件经签名散列值的二元组的集合。

可信二进制可执行文件列表中每一项的内容是可信二进制可执行文件路径及其经签名散列值、配置文件路径和动态共享库名。比如某二进制可执行文件 P 具有两个配置文件 C1 和 C2，并使用了一个动态共享库 S1，那么可信列表中关于 P 的那一行应该是：

P 的路径 P 的经签名散列值 C1 的路径 C2 的路径 S1 的文件名

可信配置文件与动态共享库列表中每一项的内容是可信配置文件或动态共享库路径及其经签名散列值，其格式与可信模块列表相同。

可信可执行脚本列表中每一项的内容是可执行脚本路径及其经签名散列值，其格式也与可信模块列表相同。

Keeper2 模块类似于 Demetrios Lambrou 方案中的 Keeper 模块，但有所改进。对于内核模块的验证，它读入可信模块列表，行为与 Keeper 类似；对于可信二进制可执行文件的验证，它读入可信二进制可执行文件列表，此时与 Keeper 不同之处在于它不仅对二进制可执行文件进行验证，而且会利用可信配置文件与动态共享库列表，对该二进制可执行文件对应的配置文件与动态共享库进行验证。

可信 shell 由 Keeper2 模块验证。它在调用脚本解释器之前对可执行脚本进行验证，然后才调用相应的解释器。

在使用这个模块中需要注意的一个问题是，虽然 Keeper2 是一个内核模块，但是在编译内核时必须把它编进内核，否则在该模块装载之前就装载的组件无法被验证。

2.4 标记文件不可修改功能的实现方法

可以通过添加一个系统调用来实现：

```
int immutable(const char *path);
```

其中 path 是指想要被标记为不可修改的那个文件，返回值表明操作是否成功。在 immutable 调用内部，可以采用引用计数的方法，分别记录文件被哪几个进程标记以及进程标记了哪几个文件。当要修改文件时查询此文件是否被进程标记(需在 Keeper2 模块中添加一个 LSM 的钩子函数)，如果标记了则拒绝修改。当进程退出时检查曾标记了哪几个文件，需要撤销这些标记信息。

2.5 修改后的 Linux 启动过程

如图 3 所示，经过以上的修改之后，系统启动过程如下：

(1)CRTM 验证支持可信计算的 OS 装载器，即 trusted Grub，验证通过之后把执行权交给 trusted Grub；

(2)在 TPM 芯片的帮助下，trusted Grub 验证 Linux 内核，验证通过之后把执行权交给 Linux 内核；

(3)Linux 内核需要在一个用户态程序的帮助下装载内核模块。这时候 Keeper2 模块就会对这个帮助程序和这些内核模块进行验证，验证通过了才会让这个帮助程序装载内核。

(4)内核启动结束时会调用 init 来启动系统服务，这时候 Keeper2 模块首先验证 init 程序以及对应的配置文件/etc/inittab 等，验证通过了则执行 init。在 init 执行过程中，很多系统服务会被启动，每当要启动一项服务，Keeper2 模块都会该服务对应的程序及其配置文件和动态共享库进行验证，验证通过了才会启动该服务。

这样，Linux 的启动过程就符合可信计算的标准了。

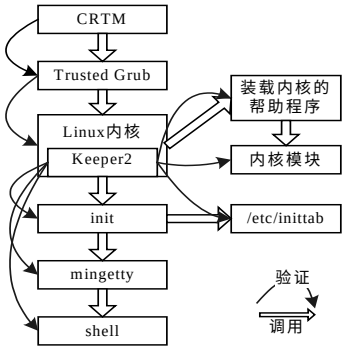

图 3 可信 Linux 的启动过程

2.6 实验结果

经过修改之后，任何内核模块、二进制可执行文件、动态共享库或配置文件的小小改变都被 Keeper2 模块发现，并根据预定的策略拒绝加载；而任何可执行脚本的改变都被修改后的可信 bash 发现，也拒绝了该可执行脚本的执行。

3 小结和未来展望

本文提出的方案大大完善了 Demetrios Lambrou 的想法，弥补了其不能验证配置文件、动态共享库和可执行脚本的缺点，保证了对所有可能改变系统可信状态的关键组件的验证。当然，本方案更加复杂，Keeper2 模块中修改文件时的钩子函数会对系统 I/O 性能造成一定的影响。今后的主要工作是对该方案进一步进行优化，以提高性能。

参考文献

1 张焕国. 可信计算平台技术研究[C]. 第 10 届全国容错计算学术会议, 北京, 2003-09.

2 Lambrou D. TCPA Enabled Open Source Platforms[Z]. http://www.crazylinux.net/downloads/projects/TCPA/TCPA_thesis.html.

3 Applied Data Security Group. Trusted Grub[Z]. http://www.prosec.rub.de/trusted_grub.html.

4 Smalley S, Fraser T, Vance C. Linux Security Modules: General Security Hooks for Linux[Z]. http://lsm.immunix.org/.

5 Trusted Computing Group. TCG Specification Architecture Overview[Z]. https://www.trustedcomputinggroup.org/home.

(上接第 168 页)

参考文献

1 Rekhter Y, Li T. A Border Gateway Protocol 4 (BGP-4)[S]. RFC 1771, 1995-03.

2 Murphy S. BGP Security Protection[Z]. IETF Internet Draft, http://draft-murphy-bgp-protect-02.txt, 2002-02.

3 Murphy S. BGP Border Gateway Protocol Security Analysis[Z]. IETF Internet Draft, http://draft-murphy-bgp-vuln-00.txt, 2001-11.

4 Kent S, Lynn C, Seo K. Secure Border Gateway Protocol[J]. IEEE Journal on Selected Areas in Communications, 2000,18(4).

5 Wan Tao, Kranakis E, Oorschot P C. Pretty Secure BGP(psBGP)[C]. Proc. of NDSS, 2005.

6 James N G. Extensions to BGP to Support Secure Origin BGP (soBGP)[Z]. IETF Internet Draft, http://Draft-ng-sobgp-bgp-extensions-02.txt.

7 Subramanian L, Roth V, Stoica I, et al. Listen and Whisper: Security Mechanisms for BGP[C]. Proc. of the 1st Symposium on Networked Systems Design and Implementation, San Francisco, CA, USA, 2004-03.

8 Kent S. Atkinson R. Security Architecture for the Internet Protocol[S]. RFC 2401, 1998-11.