UEFI Bootkit 模型与分析

唐文彬[1] 陈 熹[1] 陈嘉勇[1,2] 祝跃飞[1]
(解放军信息工程大学信息工程学院 郑州 450002)[1]
(中国科学院信息安全国家重点实验室 北京 100049)[2]

摘 要 分析了UEFI Bootkit的工作原理和关键技术;在Harold木马模型的基础上,给出了UEFI Bootkit的形式化描述;分析了UEFI Bootkit和木马在隐蔽技术方面的差异,建立了UEFI Bootkit协同隐藏的形式化模型;给出了模型的一个应用实例,理论证明了在操作系统内核启动前检测Bootkit比在操作系统启动完成后检测具有更好的效果;开发了一套在操作系统内核加载前就开始检测的UEFI Bootkit检测系统;使用检测系统进行了实际的测试,结果表明,UEFI Bootkit检测系统具有较好的检测效果,有效地验证了模型的准确性。

关键词 UEFI,形式化,Bootkit,隐蔽技术,可信计算,检测系统
中图法分类号 TP309.5 **文献标识码** A

Analysis and Detection of UEFI Bootkit

TANG Wen-bin[1] CHEN Xi[1] CHEN Jia-yong[1,2] ZHU Yue-fei[1]
(Information Engineering Institute, PLA Information Engineering University, Zhengzhou 450002, China)[1]
(State Key Laboratory of Information Security, Graduate University of Chinese Academy of Science, Beijing 100049, China)[2]

Abstract This paper analyzed the work mechanism and key technology of UEFI Bootkit, expanded the definition of Trojan according to it, illustrated the differences of hiding technology between UEFI Bootkit and Trojan, built a formal model of UEFI Bootkit cooperative concealment, showed an application of the model, proved the idea that detecting Bootkit before the operating system kernel starting can obtain a better effect than after the operating system starting. We designed and developed UEFI Bootkit detection system which works before the operating system kernel starts. The detection system was used to do practical test, and the results show UEFI Bootkit detection system obtains a good effect and has the accuracy.

Keywords UEFI, Formal description, Bootkit, Hiding technology, Trusted computing, Detection system

1 引言

随着信息技术的日益普及,恶意代码[1,2]受到了越来越广泛的关注。恶意代码技术的迅速发展,给信息安全领域不断带来新的挑战。Bootkit是恶意代码技术较为高级的阶段。可以认为,所有在开机时比Windows内核更早加载,实现内核劫持的恶意代码,都能够称为Bootkit。

关于Bootkit最早的研究源于2005年eEye Digital Security公司的研究人员发布的"BootRoot"项目[3],项目中详细介绍了如何利用Windows启动过程侵入系统内核,通过感染主引导记录(MBR)的方式,实现绕过内核检查和启动隐身。2007年Black Hat欧洲大会上,Nitin Kumar和Vipin Kumar公布了VBootKit,实现了利用启动过程入侵Vista内核。2009年,Peter Kleissner在Black Hat会议上发布Stoned Bootkit,它具有较强的通用性,能够在多个版本的Windows操作系统上运行。

随着计算机硬件的快速发展,传统的BIOS逐渐成为瓶颈,其已经很难适应现代计算机硬件的发展需求。针对传统BIOS的弊端,Intel公司提出了可扩展固件接口(Extensible Firmware Interface,EFI),以替代传统的BIOS。UEFI采用了全新的架构,其启动流程完全不同于传统的BIOS,在UEFI体系结构下,传统的Bootkit已经不再适用。

随着UEFI BIOS的普及,其安全性也逐渐受到学者的关注。Next-Generation公司的安全研究员John Heasman在2007年指出了针对EFI攻击的几种可能方法[4],但并未给出具体细节。2009年,Rafal Wojtczuk和Alexander Tereshkin指出了EFI BIOS Q45系列主板上的一个关于开机画面像素计算上的漏洞[5]。2010年,林伟[6]设计并实现了一个基于UEFI的Bootkit,该Bootkit能够感染bootmgfw.efi文件,绕过PatchGuard保护机制,在操作系统启动过程中实现内核劫持。

目前已有的基于UEFI的Bootkit分析和检测工作主要

到稿日期:2011-12-21 返修日期:2012-03-21 本文受郑州市科技创新团队(10CXTD150)资助。
唐文彬(1989—),男,硕士生,主要研究方向为信息安全;陈 熹(1988—),男,硕士生,主要研究方向为信息安全;陈嘉勇(1982—),男,博士生,主要研究方向为信息安全;祝跃飞(1962—),男,教授,博士生导师,主要研究方向为信息安全。

是针对它的实现机制进行的。本文的主要工作是通过分析已有的 UEFI Bootkit 样本，归纳总结其工作原理，给出 UEFI Bootkit 的形式化描述；分析 UEFI Bootkit 在隐藏性方面的新特点，在此基础上建立 UEFI Bootkit 的协同隐藏模型，给出了模型的一个应用实例；理论证明在操作系统内核启动前检测 Bootkit 比在操作系统启动完成后检测具有更好的效果，依据结论开发了一套在操作系统内核启动前就开始检测的 UEFI Bootkit 检测系统。通过实验验证，在操作系统内核启动前检测具有较好的效果，有效地说明了模型的准确性。

本文第 2 节分析 UEFI Bootkit 的工作原理；第 3 节给出 UEFI Bootkit 的形式化定义，建立 UEFI Bootkit 协同隐藏模型；第 4 节给出基于 UEFI 的 Bootkit 模型的一个应用实例；最后进行总结。

2 UEFI Bootkit 工作原理分析

基于 UEFI 的 Bootkit 先于操作系统内核启动，并且要在操作系统启动完成之后获得内核态权限，因此，Bootkit 必须将代码嵌入到操作系统启动流程中，并通过控制权传递与回收实现在系统启动完成后获得内核态权限的目标。Bootkit 可以在内核加载之前的所有启动组件中嵌入代码，包括 UEFI BIOS 和 OS Loader。Bootkit 传递和回收控制权主要有 Hook ExitBootServices 和逐级 Hook 两种方式。以 Windows 7 操作系统为例，分别根据两种不同的 Hook 方式分析 Bootkit 的工作原理。

2.1 越级 Hook 方式

UEFI 提供了一套完备的启动时服务(boot service)和运行时服务(runtime service)给操作系统设计者使用，ExitBootServices 便是启动时服务中的一个重要函数。当 OS Loader 把操作系统内核加载进内存之后，会调用 ExitBootServices 来停止启动时服务。然后，OS Loader 将控制权转交给内核。通过反汇编的方法，得到具体实现代码为：

```
kernel_load(kernel_image,kname,&kd,&imem,&mmem);
close_devices();
status=BS->ExitBootServices(kernel_image,cookie);
start_kernel(kd.kentry,bp);
```

从 OS Loader 转交控制权给内核的实现代码可以看到，只要对启动时服务 ExitBootServices 进行 Hook，就可以在操作系统内核执行前再次获得控制权，从而成功劫持系统内核。以在 UEFI BIOS 中嵌入代码为例，越级 Hook 方式的 Bootkit 工作原理如图 1 所示。

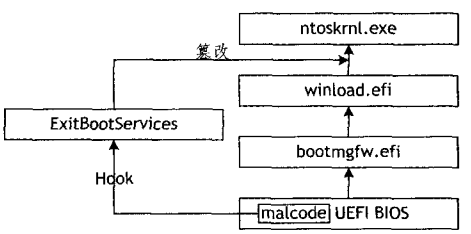

图 1 越级 Hook 工作原理

从图中可以看出，首先 Bootkit 通过刷写固件的方式将其代码嵌入到 UEFI BIOS 中，并随 BIOS 加载。Bootkit 先对 ExitBootServices 函数进行 Hook，当 Hook 完成之后将控制权重新转交给正常的启动流程，操作系统按照正常流程继续启动。启动到 winload.efi 阶段时，winload.efi 将内核加载进内存，然后调用 ExitbootServices 函数停止启动时服务，这时控制权回收到 Bootkit 中，Bootkit 可以任意篡改内核实现系统启动完成后获得内核态权限的目的。

2.2 逐级 Hook 方式

逐级 Hook 方式的 Bootkit 需要针对每一个启动组件都进行 Hook。以在 UEFI BIOS 中嵌入代码为例，Bootkit 在 bootmgfw.efi 加载进内存后对其 Hook，然后将控制权转交给正常启动流程，在 bootmgfw.efi 的 Hook 点执行时，再对 winload.efi 进行 Hook，之后将控制权再次转交给正常启动流程，当 winload.efi 的 Hook 点执行时，就可以篡改内核，达到获得内核态权限的目的。逐级 Hook 方式的 Bootkit 工作原理如图 2 所示。

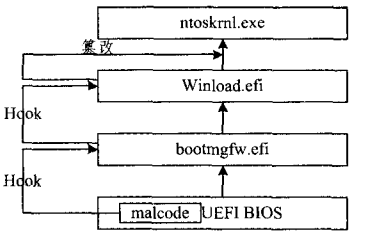

图 2 逐级 Hook 工作原理

由于逐级 Hook 需要对启动的每一个阶段都进行 Hook，因此其通用性与越级 Hook 方式相比较差。两种方式的共同点在于都需要篡改内核，来实现在操作系统启动后获得内核态权限的功能。

3 基于 UEFI 的 Bootkit 模型

3.1 木马的扩展模型

为了深入研究基于 UEFI 的 Bootkit，剖析其隐藏技术的本质，参照 Harold Thimbleby 的木马模型框架[7]，给出木马扩展模型的形式化描述。首先给出如下定义。

定义 1 操作系统内核启动后计算机的状态、程序文本和映像等统称为启动后表示，记为 R。$r\in R$，r 是有限的。

定义 2 操作系统内核启动前计算机的状态、程序文本和映像等统称为启动前表示，记为 R'。$r'\in R'$，r' 是有限的。

定义 3 $E:R\to(L\downarrow R)$，E 为表示到环境的映射。在此，通常理解为计算机的系统配置。$L\downarrow R$ 是名字到表示的部分映射，L 是名字的可数集。用户通常关心的是名字，如程序名和文件名，具体的名字属于集合 L。$L\downarrow R$ 的含义为：如果名字有定义，那么就利用名字获取其相应的表示。

定义 4 $E(r)$ 的域：names $r=\mathrm{dom}\,E(r)$，为可计算的和有限的，其中包含一些独立于 r 的名字。

定义 5 程序的含义是指程序在运行过程中完成了什么工作，用 $[\![p]\!]$ 表示。如果 $r\in R$，对于 r 中的程序 p，$[\![p]\!]$ 的含义用 $[\![\cdot]\!]:R\to(R\to R)$ 表示。程序的运行将使一种表示变化为另一种表示，如系统的状态在程序的作用下变为另一种状态。

在没有歧义的情况下，将 $\Omega=[E,R,[\cdot]]$ 用 S 表示。

定义 6 两个程序 p 和 p' 相等是指：

$\forall r\in R,[p]r=[p']r$

由于判定两个程序相等是不可计算的，因此引入程序相似关系和大多数量词 M。

定义 7 考虑检测程序输出的过程所耗费的时间，定义 R 上小于线性时间可计算的关系为相似，用～表示。相似关系是非传递的。

定义 8 如果两个程序 p 和 p' 对大多数的输入能够产生相似结果，则认为它们是不可区分的，记作 $p\approx p'$，即：

$p\approx p'$ iff $M r\in R:[p]r\sim[p']r$

为方便讨论，本文再引入一些符号。

$\hat{p}$：表示 p 被木马化后的程序。

$r\xrightarrow{l}r''$：

$r\xrightarrow{l}r''$ iff $l\in$ names r and $r''=[E(r)l]r$

可以自然扩展到 $l_1,l_2,\cdots,l_n\in L^*$ 的有限序列。

定义 9 木马关系：

$ptrojan\ \hat{p}\Leftrightarrow\exists r\in R,[p]r\not\sim[\hat{p}]r$

该定义说明如果程序 p 和 $\hat{p}$ 存在木马关系，那么至少存在一种能够将它们区分的表示。

定义 10 木马方法是一个递归的、非空的、可枚举的关系 $T\subseteq R\times R\times R'\times R'\times L$，如果 $\langle r,\hat{r},r',\hat{r}',l\rangle\in T$，则有：

$\wedge r\sim\hat{r}$

$\wedge E(r)l\approx E(\hat{r})l$

$\wedge Mt\in L^*$：

$\wedge[E(r)l]r\xrightarrow{t}r''$

$\wedge[E(\hat{r})l]\hat{r}\xrightarrow{t}\hat{r}''$

$\wedge r''\not\sim\hat{r}''$

该定义说明如果 $\langle r,\hat{r},r',\hat{r}',l\rangle\in T$ 是一个木马方法 T，操作系统启动后 $\hat{r}$ 拥有与 r 类似的环境，其中存在程序 l，对潜在的大多数输入参数来说，l 在两个环境中执行的情况是一样的，但最终会表现出不一致，那么这个 l 就是木马程序。

该模型将原来的木马定义扩展到 5 维，增加了操作系统启动前的表示。由上述模型可以看出：木马的定义重点关注于植入木马后系统环境会产生变化以及产生何种变化，而对于木马怎样使系统环境发生变化却没有涉及。同时，木马不会影响内核启动前各启动组件的状态，即木马都是在操作系统内核加载后才会执行。

3.2 基于 UEFI 的 Bootkit 的形式化描述

木马是在操作系统启动之后加载运行的，而基于 UEFI 的 Bootkit 在操作系统内核启动之前就已经获得控制权，并通过控制权传递，在系统启动完成后执行木马的功能。也就是说，在操作系统启动完成后 Bootkit 可以使用木马模型进行刻画，此时只需在操作系统内核启动之前对其进行定义即可。

定义 11 计算机程序 p 由程序片段 $a_1,a_2,\cdots,a_n$ 组成，即 $p=\{a_1,a_2,\cdots,a_n\}$。

定义 12 计算机操作系统启动平台用 S 表示，平台由各个组件组成，一个组件可能由更小的组件组成，最小规模的组件可以是一段可执行的程序代码。Bootkit 化后的操作系统启动平台用 $\hat{S}$ 表示。

定义 13 控制权传递关系可以用一个二元关系表示：$\rightarrow\in S\times S,A_i\rightarrow A_j$ 表示组件 A_i 将控制权传递给 A_j，其中 A_n 表示操作系统内核获得控制权。因此控制权传递可以用下面的式子表示：

$A_1\rightarrow A_2\rightarrow\cdots\rightarrow A_n,A_i\subset S,A_i$ 启动状态的表示为 r_i'

如果两个表示 r 和 $\hat{r}$ 是木马关系而且在操作系统启动之前就获得执行，并具有传递和回收控制权的功能，那么 $r,\hat{r}$ 是名字 l 上的 Bootkit 关系。其具体定义如下。

定义 14 Bootkit 方法是一个木马方法 $V\subseteq R\times R\times R'\times R'\times L$ 并满足附加的条件，如果 $\langle r,\hat{r},r',\hat{r}',l\rangle\in V$ 则有：

$\wedge r\sim\hat{r}$

$\wedge E(r)l\approx E(\hat{r})l$

$\wedge Mt\in L^*$：

$\wedge[E(r)l]r\xrightarrow{t}r''$

$\wedge[E(\hat{r})l]\hat{r}\xrightarrow{t}\hat{r}''$

$\wedge r''\not\sim\hat{r}''$

$\wedge a_0\in l$

$\exists i\in[1,n)$：

$\wedge A_i\rightarrow a_0$

$\wedge Mt\in L^*$：

$\wedge[E(r_n')l]r_n'\xrightarrow{t}r_n''$

$\wedge[E(\hat{r}_n')l]\hat{r}_n\xrightarrow{t}\hat{r}_n''$

$\wedge r_n''\not\sim\hat{r}_n''$

该定义说明基于 UEFI 的 Bootkit 在操作系统启动之后其实质是一个木马程序。同时，Bootkit 程序 l 将程序片段 a_0 嵌入到操作系统启动过程中执行，即 Bootkit 在操作系统内核启动前就获得执行权。在内核启动前对于大多数输入来说，Bootkit 在两个环境的执行情况是一样的，但是最终会在操作系统内核获得控制权时表现出不一致。

3.3 基于 UEFI 的 Bootkit 隐藏技术

传统的木马在植入目标系统之后，会采取各种技术隐藏自身，以避免被发现。张新宇等人在文献[8]中提出了木马协同隐藏的思想，并将木马的隐藏技术分为 3 类，分别为本地隐藏、通信隐藏和协同隐藏，较好地总结了现有的木马隐藏技术。但是，对于协同隐藏中一个重要的因素－启动隐藏，文献中却并未涉及，同时由于 Bootkit 技术在当时还并未出现，因此文献[8]对文件隐藏的概括还不够全面。作者在文献[8]的基础上，总结了木马的启动隐藏技术，对比了 Bootkit 和木马在启动隐藏和文件隐藏的差异，给出了 Bootkit 协同隐藏的形式化描述。

启动隐藏。1)修改注册表启动项,将自身设置为启动程序。2)修改系统服务,设置为通过服务加载。3)感染正常的启动程序,如 explorer. exe,随正常程序启动。木马的启动时机一般滞后于操作系统内核,被操作系统内核直接或间接加载执行。

Bootkit 在操作系统启动之后可以用木马刻画,因此,其大部分隐藏技术都与木马相似。它与木马隐藏技术的差异主要表现在启动隐藏和文件隐藏方面,如表 1 所列。从表中可以看到,在启动隐藏和文件隐藏方面,Bootkit 比木马具有更高的隐蔽性。

表 1 Bootit 与木马隐藏技术对比

	木马	Bootkit
启动隐藏	修改注册表,设置为自启动或服务启动,启动时机滞后于操作系统内核	插入或篡改启动组件如 UEFI BIOS,启动时机先于操作系统内核
文件隐藏	一般通过修改系统 API 的正常功能实现隐藏,会在文件系统下驻留文件	存放在固件或 EFI 分区中,在文件系统下一般不留痕迹

参考张新宇的木马协同隐藏模型,对 UEFI 下 Bootkit 的协同隐藏进行形式化描述。以$\hat{p}$表示 Bootkit 的主程序,在操作系统启动完成后包含$\hat{u}_1,\hat{u}_2,\cdots,\hat{u}_k$若干个子程序,因此$\hat{u}_1,\hat{u}_2,\cdots,\hat{u}_k$都具有木马的属性。在操作系统下,每个子程序$\hat{u}_i$($i\in[1,k]$)隐藏某一类属性,它们共同作用隐蔽$\hat{p}$的特征。

设$r,r',r'',\hat{r},\hat{r}',\hat{r}'',r_i,r_i',\hat{r}_i,\hat{r}_i',r_i'',\hat{r}_i''\in R,\hat{u}_i,i\in[1,k]$,$m$为$\hat{p}$属性的类数,$k\geqslant m$,则

$$\wedge r_i\sim\hat{r}_i \tag{1}$$

$$\wedge E(r_i)\hat{u}_i\approx E(\hat{r}_i)\hat{u}_i \tag{2}$$

$$\wedge Mt\in L^*\ \wedge[E(r_i)\hat{u}_i]r_i\xrightarrow{t}r_i''\ \wedge[E(\hat{r}_i)\hat{u}_i]\hat{r}_i\xrightarrow{t}\hat{r}_i''\ \wedge r_i''\not\sim\hat{r}_i'' \tag{3}$$

$$\wedge r\sim\hat{r} \tag{4}$$

$$\wedge\hat{u}_i\in L^* \tag{5}$$

$$\wedge[E(r)\hat{p}]r\xrightarrow{\hat{u}_1,\hat{u}_2,\cdots,\hat{u}_i}r''\ \wedge[E(\hat{r})\hat{p}]\hat{r}\xrightarrow{\hat{u}_1,\hat{u}_2,\cdots,\hat{u}_i}\hat{r}''$$

$$\begin{cases}\wedge r''\not\sim\hat{r}'', & 1\leqslant i<m\\ \wedge r''\sim\hat{r}'', & i\geqslant m\end{cases}\ \hat{u}_i\text{ 协同隐藏 }\hat{p} \tag{6}$$

$$\wedge a_0\in l \tag{7}$$

$$\exists i\in[1,n):\ \wedge A_i\to a_0 \tag{8}$$

$$\wedge Mt\in L^*\ \wedge[E(r_j')\hat{p}]r_j'\xrightarrow{t}r_j''\ \wedge[E(\hat{r}_j')\hat{p}]\hat{r}_j'\xrightarrow{t}\hat{r}_j''$$

$$\begin{cases}\wedge r_j''\sim\hat{r}_j''\text{ or }\wedge r_j''\not\sim\hat{r}_j'', & 1\leqslant j<n\\ \wedge r_j''\not\sim\hat{r}_j'', & j=n\end{cases} \tag{9}$$

其中式(1)—式(6)表示 Bootkit 在操作系统启动完成后的协同隐藏,即木马的协同隐藏;式(7)和式(8)表示 Bootkit 在操作系统内核启动前就获得控制权;式(9)表示 Bootkit 在操作系统内核启动前的协同隐藏。

由 Bootkit 协同隐藏的形式化描述可以看出,在操作系统启动完成后,如果子程序可以隐藏 Bootkit 主程序具有的所有属性即 $i\geqslant m$,那么检测程序就无法将$\hat{p}$与正常程序 p 区别开来。在操作系统内核加载之前,对于大多数输入参数而言,检测程序能以一定的概率将$\hat{p}$与正常程序 p 区别开来,但是当操作系统内核获得控制权时则会以概率 1 表现出不一致。

4 基于 UEFI 的 Bootkit 模型的应用

基于 UEFI 的 Bootkit 在操作系统启动完成后一般都具有内核态的权限,为其隐藏提供了便利,因此具有较强的隐蔽性,一般的检测程序都难以将其检测出来。同时,Bootkit 先于检测程序启动,可以直接破坏检测程序正常功能的执行,实现"隐身"。

4.1 Bootkit 检测判定定理

目前,对于 Bootkit 检测大多都是在启动之后,由于 Bootkit 在启动后存在协同隐藏,可能导致检测程序无法将其与正常程序区别开来,检测效果不好。于是,在 UEFI 的 Bootkit 模型的基础上,本文给出基于 UEFI 的 Bootkit 检测判定定理。

定理 1 Bootkit 在操作系统启动完成后的检出率为 α,在操作系统内核启动前的检出率为 β,则 $\alpha\leqslant\beta$。

证明:1)如果 l 为 Bootkit 主程序,且能够在操作系统启动完成后被检出,则 l 在操作系统内核启动前能够被检出。

$\forall l$,l 是 Bootkit 主程序,且 l 在操作系统下能够被检测程序检出,那么根据协同隐藏模型的式(6)有:

$$\wedge\hat{u}_i\in L^*\ \wedge[E(r)l]r\xrightarrow{\hat{u}_i}r''\ \wedge[E(\hat{r})l]\hat{r}\xrightarrow{\hat{u}_i}\hat{r}''\ \wedge r''\not\sim\hat{r}''$$

同时,根据式(9),在操作系统内核启动前有:

$$\wedge Mt\in L^*\ \wedge[E(r_j')l]r_j'\xrightarrow{t}r_j''\ \wedge[E(\hat{r}_j')l]\hat{r}_j'\xrightarrow{t}\hat{r}_j''\ \wedge r_n''\not\sim\hat{r}_n''$$

即 l 也能在操作系统内核启动前被检测程序检出。

2)存在 l 为 Bootkit 主程序,不能够在操作系统启动完成后被检出,但是 l 在操作系统内核启动前能够被检出。

l 是 Bootkit 主程序,取 $i=m$,根据式(6),在操作系统启动完成后有:

$$\wedge\hat{u}_i\in L^*$$

$$\wedge[E(r)l]r \xrightarrow{\hat{u}_i} r''$$

$$\wedge[E(\hat{r})l]\hat{r} \xrightarrow{\hat{u}_i} \hat{r}''$$

$$\wedge r'' \sim \hat{r}''$$

即 l 在操作系统下不能被检测程序检出。但是根据式(9),在操作系统内核启动前有:

$$\wedge Mt \in L^*$$

$$\wedge[E(r_j')l]r_j' \xrightarrow{t} r_j''$$

$$\wedge[E(\hat{r}_j')l]\hat{r}_j' \xrightarrow{t} \hat{r}_j''$$

$$\wedge r_n'' \neq \hat{r}_n''$$

即 l 能在操作系统内核启动前被检测程序检出。

综合 1)和 2),有 $\alpha \leqslant \beta$,证毕。

由 Bootkit 协同隐藏模型可以看到,即使 Bootkit 采用协同隐藏技术进行隐藏,在操作系统内核获得控制权时还是会表现出不一致,其一定能被检出,即 $\beta=1$。

4.2 基于 UEFI 的 Bootkit 检测实验与结果分析

由定理 1 可知,在操作系统内核启动前检测 Bootkit 将具有更好的效果。因此,采用在操作系统内核启动前进行检测的方法,参考可信计算的思想[9-11]开发了一个基于完整性校验的 UEFI Bootkit 检测系统。检测系统存储在光盘中,该系统能够在内核启动前对各个启动组件进行完整性校验,系统整体框架如图 3 所示。

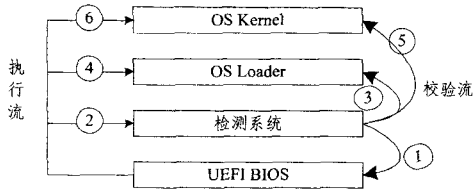

图 3 检测系统整体框架

由图 3 可以看出,②、④、⑥为系统启动的执行流程,即控制权传递过程为:UEFIBIOS→检测系统→OSLoader→OS Kernel。①、③、⑤为系统的校验流,依次对各个启动组件进行可信度量。具体的过程为:首先,设置启动选项为光盘启动,则 UEFI BIOS 执行完成之后将自动加载检测系统。这时,检测系统加载 OS Loader,并判断 digist(detect,OS Loader)=expect(OS Loader) 是否成立,如果不成立,则系统被感染 Bootkit;如果成立,则 Hook OS Loader,并将控制权转交给 OS Loader。当 OS Loader 加载完 OS Kernel,执行到 Hook 点时,代码执行权限回收到检测系统中,检测系统判断 digist(detect,OS Kernel)=expect(OS Kernel)是否成立,如果不成立,则系统被感染 Bootkit;如果成立,则系统未感染 Bootkit。其中 digist(A,B)表示组件 A 对组件 B 进行摘要运算的结果,detect 表示检测系统,expect(A)表示组件 A 哈希的预期值。

由基于 UEFI 的 Bootkit 模型可知,对于感染 Bootkit 的主机,在操作系统内核获得控制权时一定会表现出不一致,即必然会存在 digist(detect,A_n)≠expect(A_n),因此,一定可以被检测系统检出。利用 Bootkit 检测系统分别对 UEFI Bootkit 进行检测,结果如表 2 所列。其中 desert.exe[12] 是插入到 UEFI BIOS 中的 Bootkit,采用的是越级 Hook 方式;cylon.exe[6] 也是插入到 UEFI BIOS 中的 Bootkit,采用的是逐级 Hook 方式;linbot.exe[6] 是插入到 bootmgfw.efi 中的 Bootkit,采用的是逐级 Hook 方式。从表中可以看到,开发的检测系统能够有效地检测 Bootkit,验证了基于 UEFI 的 Bootkit 模型的准确性。

表 2 UEFI Bootkit 检测结果

样本名称	代码插入位置	Hook 方式	检测结果
desert.exe	UEFI BIOS	越级 Hook	检测成功
cylon.exe	UEFI BIOS	逐级 Hook	检测成功
linbot.exe	bootmgfw.efi	逐级 Hook	检测成功

结束语 本文基于 UEFI Bootkit 的原理,分析了 UEFI Bootkit实现的关键技术。然后借鉴 Harold Thimbleby 的木马模型,给出了 UEFI Bootkit 定义的形式化描述。在 UEFI Bootkit 定义的基础上,讨论了 UEFI Bootkit 和木马在隐蔽技术上的差异,建立了 UEFI Bootkit 协同隐藏模型,并给出了模型的一个应用,证明了在操作系统内核启动前检测 Bootkit 比操作系统启动完成后检测具有更好的效果。根据定理开发了一套在操作系统内核启动前就开始检测的 UEFI Bootkit 检测系统,实验结果表明,系统具有较好的检测效果,有效地验证了模型的准确性。

参 考 文 献

[1] Cadar C, Genesh V, Pawlowski P M, et al. EXE: Automatically generating inputs of death[C]//CCS'06 Proceedings of the 13th ACM Conference on Computer and Communications Security. 2006:322-335

[2] Levine J F, Grizzard J B, Owen H L. Detecting and categorizing kernel—level rootkits to aid future detection[J]. IEEE Security and Privacy, 2006, 4(1):27-32

[3] Soeder D, Permeh R. Eeye BootRoot[EB/OL]. http://research.eeye.com/html/tools/RT20060801-7.html, 2005-09

[4] Heasman J. Implementing and Detecting an ACPI BIOS Rootkit [EB/OL]. http://www.blackhat.com/presentations/bu-usa-07/Heasman/Presentation/bh-usa-07Heasman.pdf, 2007-02

[5] Wojtczuk R, Tereshkin A. Attacking Intel BIOS[EB/OL]. http://www.blackhat.com/presentations/bh-usa-09/WOJTCZUK/BHUSA09-Wojtczuk-AtkIntelBios-SLIDES.pdf, 2009-08

[6] 林伟. 基于 UEFI BIOS 的 Bootkit 技术研究[D]. 郑州:信息工程大学,2010

[7] Thimbleby H, Anderson S, Cairns P. A framework for medelling trojans and computer virus infection[J]. The Computer Journal, 1998, 41(7):444-458

[8] 张新宇,卿斯汉,马恒太,等. 特洛伊木马隐藏技术研究[J]. 通信学报,2004,25(7):153-159

[9] 沈昌祥,张焕国,冯登国. 信息安全综述[J]. 中国科学 E 辑:信息科学,2007,37(1):129-150

[10] 冯登国,秦宇,汪丹,等. 可信计算技术研究[J]. 计算机研究与发展,2011,48(8):1332-1349

[11] 黄强. 基于可信计算的终端安全体系结构研究[D]. 武汉:海军工程大学,2007

[12] 朱瑜. Bootkit 检测技术研究[D]. 郑州:信息工程大学,2010