一种 Web 服务器安全机制的实现方法

陈伟东[1]，刘 刚[1]，徐 峥[1]，耿坤英[2]
(1. 浪潮嘉信信息技术有限公司，北京 100085；
2. 清华大学 软件学院，北京 100083)

摘 要：针对 Web 网站点的入侵事件不断发生，现有的防火墙、IDS 等设备都不能有效防止入侵者篡改网站中的网页、盗取重要信息等攻击，提出了 Web 服务器安全较完备机制，从核心层保证 Web 站点中的网页不会被黑客篡改，恶意代码在系统中不会肆意发作。该机制重构了操作系统核心层权限访问控制模型，对操作系统文件、注册表、进程和网络等资源采用白名单规则，并采用多机制相结合的方式提高 Web 服务器的抗攻击能力。

关键词：服务器安全；系统保护；文件驱动；网络驱动
中图分类号：TP393.08 文献标识码：A 文章编号：1674-7720(2012)02-0012-04

Web server security and protect's new methods

Chen Weidong[1], Liu Gang[1], Xu Zheng[1], Geng Kunying[2]
(1. Inspur Jiaxin Computer Infotech Co. Ltd, Beijing 100085, China;
2. Software School, Tsinghua University, Beijing 100083, China)

Abstract: With the development of Internet, The attacks to Web site incidents continue to occur, the existing firewall, IDS and other equipment can not effectively prevent intruders tampering website pages and other files, steal important information such attacks. This paper presents a more complete Web server security mechanism, the core layer of assurance from the Web site pages tampering by hackers, malicious code in the system will not wantonly attack. The mechanism of reconstruction of the operating system kernel level privileges to access control model, operating system files, registry, process, and network resources such as the use of white list rules, and using a combination of multiple mechanisms to increase resistance to attack Web servers.

Key words: Web; server security; system protect; network driver; file system driver

随着互联网技术的飞速发展，作为信息系统的核心设备，服务器的安全提升到了一个新的高度。服务器中存储和处理的大量核心业务数据，其安全性愈显重要，传统防病毒系统、防火墙、IDS 等设备无法保证服务器系统高度保密和信息完整性要求。多核服务器的安全保障难度再增高，采用新的安全机制来提供 Web 服务器抵抗黑客和恶意代码攻击尤为重要。Web 应用必须有 80 和 443 端口，恶意用户正是利用了这两个端口执行各种恶意操作：偷窃、操控、破坏 Web 应用中重要信息。

为透明提升 Web 服务器操作系统安全等级，最大程度地保证系统兼容性，本文通过对文件系统和网络底层架构的理解，结合对规则的成功运用，实现了操作系统安全和网络安全的有效结合。

1 原理与架构

系统分为驱动层和应用层。驱动层包括一个文件过滤驱动程序和一个 NDIS 网络防火墙驱动；应用层包括一个应用程序和与驱动交互的 DLL。应用层采用了 WTL 做界面，在驱动程序启动时，读取相应的规则文件，包括文件规则(所有进程对文件读写的规则和特殊进程对文件读写的规则)。文件读写权限有禁用、只读、正常读写等。

计算机病毒发作时的行为一般有写注册表项、生成文件、远程线程注入等。安全防护驱动拦截系统服务调用，通过分析例程调用的参数得到文件、进程等相关信息。系统服务调度表(SSDT)保存着本地系统服务的地址，可以定位函数的内存地址。

文件系统过滤驱动对 Native API 做截获，对 IoCreateFile 做 INLINE HOOK，在文件读写时根据函数参数和例程上下文信息得到相关文件全路径和进程相关信息。与加载到内存中的文件规则做判断。规则加载方法是开启一个系统线程，在系统线程循环中判断规则文件是否

被修改。如果被修改,则重新加载规则文件到内存。

对注册表的防护,则是 HOOK 了 ZwCreateKey、ZwDeleteKey、ZwEnumerateKey、ZwOpenKey 等函数。在函数内部得到操作的进程全路径和注册路径,与规则文件中的进程名和注册表路径比较,如果符合规则,则采取规则文件中的动作对注册表操作控制,文件动作规则包括禁用、只读结合进程名的对比,只有进程名和文件名与规则中的进程名和文件名完全相同时,执行规则内相应的禁用和只读动作等。

进程和进程保护采用对 ZwOpenProcess 的 HOOK 方法实现。在截获的 ZwOpenProcess 例程内,得到进程 ID。把进程 ID 与内存规则链表中的保护进程的 ID 作对比。如果是需要保护进程的 ID,则返回无效句柄。进程注入多数采用 CreateRemoteThread 方法,在应用层采用 WriteProcessMemory 函数。在核心层则是 NtCreateThread 例程和 ZwWriteVirtualMemory 例程。防止复制句柄则是在驱动层截获 ZwDuplicateObject 例程。进程保护的最低层可采用 KiInsertQueueApc 方法的截获来实现。目前较为高级的进程保护都采用 KiInsertQueueApc 方法实现,如 XueTr 等。图 1 为服务器 Web 安全系统序列图。

对 64 位操作系统 (Windows Vista 和 Windows 2008 Server)等,由于操作系统采取了对内核保护的措施(PatchGuard 技术),其内核被锁定了,任何第三方软件都无法对其进行修改。SSDT HOOK 在 64 位操作系统上失效。

采用微软 WDK 开发包提供 minifilter 框架,对文件权限的修改在 IRP_MJ_CREATE 内进行截获,对于只读文件,如果规则比较发现其是需要进行权限修改的只读文件,有 WRITE 的标记,则修改为 GENERIC_READ 标记。对需要禁用的文件,则在 IRP_MJ_CREATE 的后处理(PostCreate) 例程内采用 FltCancelFileOpen 对相应的文件禁用。

在网络安全方面采用 NDIS 低层技术构建网络防火墙,提供网络层访问控制和攻击保护服务。统一部署在 Web 应用的前端。网络防火墙是整个 Web 应用安全体系结构的一个组件,防御网络层黑客攻击(网络扫描、telnet 等),阻止蠕虫的传播。

2 文件和注册表访问权限研究与实现

文件访问控制采用 Inline Hook 方法,对 IoCreateFile 做截获。文件规则包括文件全路径名、打开文件的进程名、访问权限(只读、禁用、正常使用等)等多元参数组。在截获函数内,首先根据参数和上下文得到文件全路径名。然后得到进程名。与内存规则内的文件和进程名对比,如果符合则根据规则中的动作对文件只读、禁用等操作。如果不符合,则采取默认动作。驱动内部需要得到保护进程的列表和进程 ID 等。进程加载通知(PsSetCreateProcessNotifyRoutine);DLL 或驱动加载通知(PsSetLoadImageNotifyRoutine);低层驱动维护着活动进程链表和文件、进程、注册表规则链表等。图 2 为文件网络驱动与应用层关系序列图。

Web 服务器的安全需要考虑到 Web 服务应用程序的安全,包括远程线程的防止注入、复制句柄、远程内存读写、子进程创建的预防和判断等。远程线程注入通过 CreateRemoteThread 实现,需要调用 WriteVirtualMemory 等函数,在内核层则是调用 ZwWriteVirtualMemory。ZwWriteVirtualMemory 是在 NTDLL.DLL 中导出的,在驱动内得到 NTDLL.DLL 的地址,然后得到相应导出函数 ZwWriteVirtualMemory 地址。截获地址后,在截获函数内判断进程 ID 号是否是在进程保护列表中的进程,如是则返回 STATUS_ACCESS_DENIED。

驱动底层的难点主要有文件只读、禁用规则,结合进程规则的对比,以及规则文件的设计和在内存中与进程和文件名的对比。驱动层原来对文件对比采用 SSDT HOOK 方法,拦截 ZwOpenfile 和 ZwCreateFile 函数,后来改用了一个函数,采用了 Inline Hook 加汇编的方式,简

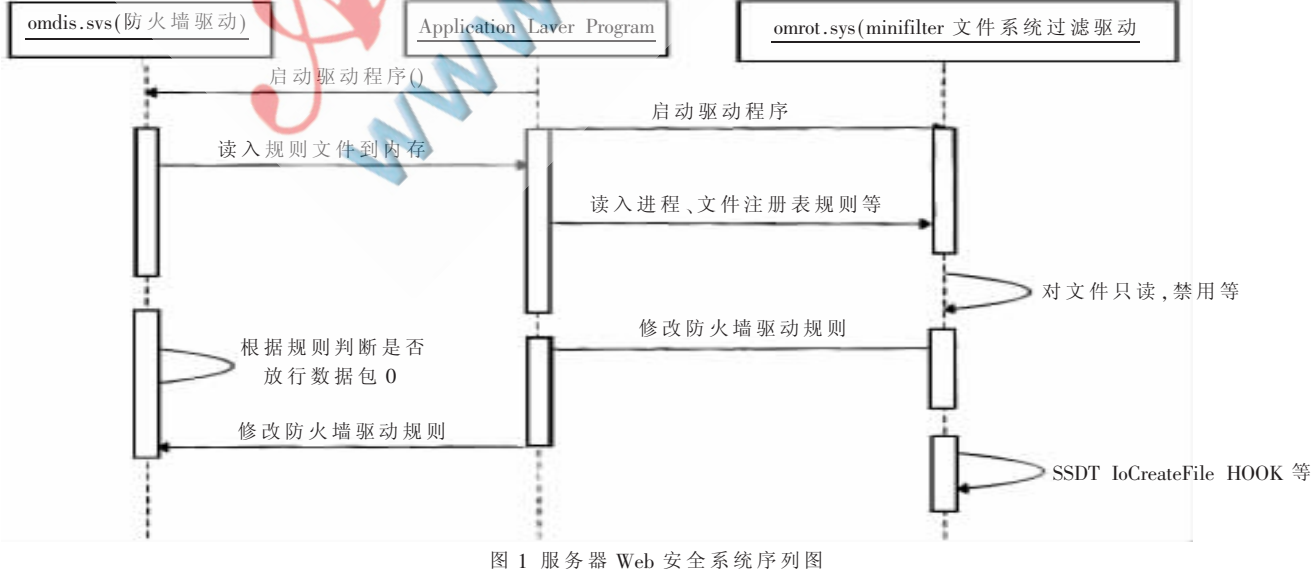

图 1 服务器 Web 安全系统序列图

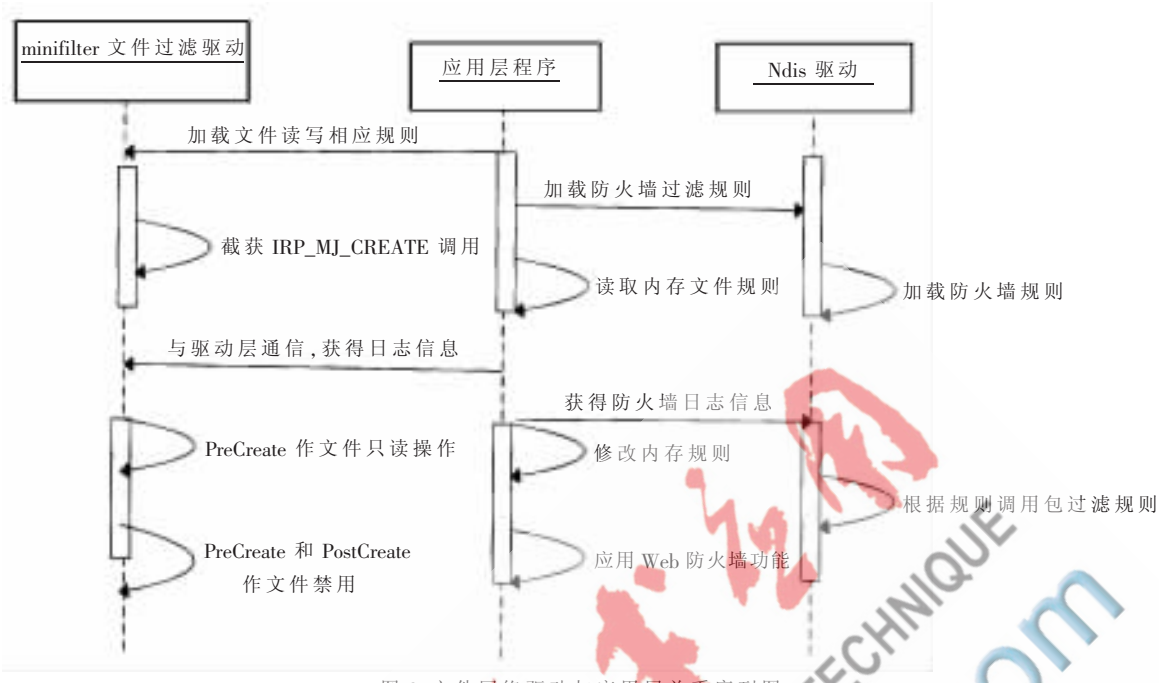

图 2 文件网络驱动与应用层关系序列图

化了对函数的比较,收到了比较好的效果。对注册表的规则和禁用,采用了 Regmon 中的方法,在注册表相应的函数内得到注册表的访问全名,与注册表规则进行比较,采取相应的禁用等方法。

规则文件加上只读等控制后,对规则的判断首先需要对进程名进行判断,是否符合规则中的进程名。判断符合后,再对文件名进行规则判断,如果文件名相同,则按规则文件中数据结构对文件做只读和禁用等对复制句柄的禁止。

在 64 位服务器系统上,需要对驱动进行签名。Minifilter 架构可在 32 位和 64 位操作系统上运行。可运行的系统包括 WindowsXP、Windows 2003、Windows2008 Server 64 位等。在 64 位多核系统中需要调整一些不兼容的函数。

驱动难点还有在多核服务器上的驱动例程,如自旋锁(SPIN LOCK)的获取等,遇到了数次 BSOD.都是多核锁造成的,多核下要用 KeAcquireInStackQueuedSpinLock.如果不想改动 IRQL,只能用资源锁(ERESOURCE)了。

3 服务器 Web 安全数据结构和接口方案

用户接口主要是应用层的程序与驱动进行 DeviceIoControl 交互,应用程序启动时,需要与驱动程序会话,调用相应的函数。还有应用层需要定时与驱动层查询得到相关的日志。

```
//驱动层与应用层应答 IoControl
#define IOCTL_REPLY_USERLAND_REQUEST
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x802, METHOD_
    BUFFERED,FILE_ANY_ACCESS)
//对规则文件加密和解密 IoControl
#define IOCTL_RULEFILE_ENCRYPT
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x805, METHOD_
    BUFFERED, FILE_ANY_ACCESS)
//得到系统运行日志 IoControl
#define IOCTL_DPROTECT_GETLOG
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x809, METHOD_
    BUFFERED, FILE_ANY_ACCESS)
//应用层与驱动层通信 EVENT
#define IOCTL_REG_EVENT
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x800, METHOD_
    BUFFERED, FILE_ANY_ACCESS)
//日志记录数据结构
typedef struct _LOG_RECORD {
    ULONG       Length;
    ULONG       SequenceNumber;
    RULE_TYPE   LogType;
    ULONG       uWhat;
    LARGE_INTEGER  origTime;
    CHAR        Name[0];
} LOG_RECORD, *PLOG_RECORD;
//规则类型
typedef enum _RuleType
{
    RULE_FILE = 0,                  //文件规则类型
    RULE_DIRECTORY,                 //文件路径类型
    RULE_REGISTRY,                  //注册表规则
    RULE_SECTION,
    RULE_PROTECT,                   //需要保护的进程
    RULE_REGPROCESS,  //与注册表规则对应的进程
    RULE_FILEPROCESS,   //与文件规则对应的进程
    RULE_NETWORK,
```

```
    }RULE_TYPE;
//进程规则结构
typedef struct _PROCRULE
{
    ULONG   nId;
    CHAR    procName[PROCNAMELEN];
    short nRuleType;
    DWORD  Reserved;
} PROCRULE, *PPROCRULE;
//注册表规则
typedef struct _REGRULE
{
    ULONG nId;
    CHAR  procName[PROCNAMELEN];
    short nRuleType;
} REGRULE, *PREGRULE;
//注册表进程规则
typedef struct _REGPROCRULE
{
    ULONG   nId;
    CHAR  procName[PROCNAMELEN];
    short nRuleType;
} REGPROCRULE, *PREGPROCRULE;
//文件规则数据结构
typedef struct _FILERULE
{
    ULONG   nId;
    ULONG   nReadWrite;
    CHAR    procName[PROCNAMELEN];
    short nRuleType;
    BOOLEAN    bDirectory;
} FILERULE, PFILERULE;
//过滤规则结构
typedef struct _FLTRULE
{
    struct FLTRULE    *next;
    int   ruleLine;
    int   nAction;
    BOOLEAN bLog;
    enum RULE_TYPE ruleType;
    union {
        struct _PROCRULE   procRule;
        struct _REGRULE    regRule;
        struct _FILERULE   fileRule;      //文件规则
        struct _PROCRULE   protectRule;
                          //禁止用任务管理器结束进程
        struct _REGPROCRULE   regProcrule;
    }Rule;
} FLTRULE, *PFLTRULE;
```

Minifilter 应用层和驱动层的通信方案是在应用层采用 FilterConnectCommunicationPort 建立通信端口的连接。然后初始化相关参数数据，创建日志查询线程。用 FilterSendMessage 例程查询驱动内日志信息。

经过测试人员详细测试和客户现场应用，本系统达到了良好的应用效果，从驱动层到应用层都运行良好。目前系统支持 32 位和 64 位服务器系统，对服务器安全要求较高的企事业单位有比较好的保障作用。

参考文献

[1] 耿玉波，夏鲁宁，杜皎，等.一种 Web 服务器安全机制的研究与实现[J].计算机工程，2006(11):189-191.

[2] NAGAR R. Windows NT file system internals[EB/OL]. [2007-04-01].http://download.csdn.net/source/168266.

[3] RUSSINOVICH M E, SOLOMON D A. Microsoft windows Internals. Fourth Edition[M]. Microsoft Press, 2007.

[4] 汤方澄，杨小虎，董金祥.基于智能 Agent 的网络安全监控系统的研究[J].计算机工程，2002，28(12):63-65.

(收稿日期:2011-10-31)

作者简介：

陈伟东，男，1970 年生，高级程序员，系统分析师，主要研究方向：网络安全。