基于UEFI固件的BOOTKIT检测技术研究

刘文祺[1]，范明钰[1]，田 伟[2]，王光卫[1]

(1. 电子科技大学计算机科学与工程学院 成都 611731；2. 中国人民解放军95333部队 长沙 410114)

【摘要】基于UFEI的BOOTKIT攻击能对UFEI固件、操作系统的完整性进行破坏，严重影响计算机安全。基于此，该文提出一种新的基于UEFI固件的BOOTKIT检测方法UDS，来保护固件和操作系统的安全。UDS以UEFI虚拟设备驱动程序的形式实现，在OS之前加载启动；采用了将完整性检测与文件恢复相结合的策略，对固件和操作系统内核进行保护；并通过代码混淆和文件隐藏的方法，防止UDS自身被BOOTKIT攻击。实验表明，UDS能有效保护固件和OS的完整性，防范基于UEFI的BOOTKIT攻击，具有启动时间早、空间开销少及自我保护性好的优点。

关 键 词 BOOTKIT; 文件恢复; 完整性检测; 自我保护; 统一可扩展固件接口

中图分类号 TP309 **文献标志码** A doi:10.3969/j.issn.1001-0548.2018.06.016

BOOTKIT Detection Based on UEFI

LIU Wen-qi[1], FAN Ming-yu[1], TIAN Wei[2], and WANG Guang-wei[1]

(1. School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 611731;
2. The 95333 troop of People's Liberation Army of China Changsha 410114)

Abstract UEFI-based BOOTKIT compromise the integrity of both UEFI firmware and OS, posing a fatal threat on the security of computer. In response, a new bootkit defense system based on UEFI, named UDS, is proposed in this paper. The proposed UDS is implemented as a UEFI device driver, which is booted before the OS. By adopting a strategy that combines integrity checking and file restoring, UDS protects firmware and OS kernels. And the methods of code obfuscation and file hiding are introduced to prevent UDS itself from being attacked by BOOTKIT. Finally, several experiments had been conducted to prove that UDS can protect itself from the attack of BOOTKIT, while effectively protecting integrity of both OS and firmware.

Key words BOOTKIT; file recovery; integrity checking; self-protection; UEFI

UEFI是操作系统与固件平台间新的接口规范。UEFI使用高级语言开发，采用模块化构建方式，功能丰富，扩展性良好[1]。随着UEFI固件普及，出现了针对UEFI固件的BOOTKIT攻击。文献[2]通过固件SMM漏洞向UEFI固件植入恶意代码，攻击TXT (trusted execution technology)安全机制[3]。2015年，文献[4]对UEFI固件进行漏洞攻击，使数以百万的计算机无法正常使用。

UEFI固件BOOTKIT隐藏在固件中，于操作系统启动前执行，通过篡改UEFI固件文件以及操作系统(OS)核心文件，实现攻击目的[4]。基于UEFI的BOOTKIT的防范方法，其最终目的是确保UEFI固件和OS核心文件的完整性。现有完整性检测方法主要分为两类：一是固件完整性检测；二是OS文件检测。文献[5]采用代码特征对比的策略，提出了一种基于UEFI固件的恶意代码检测机制，但大量的病毒特征码给UEFI固件内存造成了沉重负担。文献[6]以TPM芯片为可信根，建立了一条系统启动的可信链来确保固件完整；该算法实现复杂、代码量大，且容易被BOOTKIT绕过，存在实用性低的问题。Copilot[7]利用PCI卡周期性地对OS内核进行访问以检测内核代码的完整性；改进的Copilot[7]则能检测OS内核中数据的完整性。Servicsor[8]可以在OS的整个运行周期中保障可执行程序的合法性。这些检测方法能有效检测OS内核的完整性，但是不能独立于OS运行，也不能对固件文件进行完整性检测。文献[9]提出了在执行UEFI固件可执行文件前验证其数字签名的方法，该方法原理简单，能有效检测文件的完整性，但是不能对异常文件进行恢复。

针对上述不足，本文提出一种新的UEFI固件BOOTKIT检测方法UDS(UEFI detection system)，其主要特点有：1) 以UEFI虚拟设备驱动程序的形式实

收稿日期：2017－05－05；修回日期：2017－11－21
基金项目：国家自然科学基金(60373109，60272091)
作者简介：刘文祺(1991－)，男，博士生，主要从事信息安全、计算机操作系统方面的研究.

现[1]，能独立于OS运行；2) 采用将完整性检测与文件恢复相结合的策略，分别在DXE阶段和OS BOOT阶段对固件、OS文件进行完整性检测，并对异常文件进行恢复；3) 能自我保护，防止BOOTKIT对自身进行攻击；4) 小巧简便，利用UEFI固件接口和UEFI隐藏分区开发，减少UDS占用的ROM空间。

最后实验表明，UDS能在操作系统启动之前，成功防范固件攻击并恢复异常文件，具有启动早、开销少，能自我防护的优点。

1 UDS设计

根据SPEC[1]，UFEI的启动阶段按先后次序被分为：PEI、DXE、BSD、TSL、RT。UEFI内核和UEFI基础服务，如：文件系统、网络I/O以及认证等，主要在DXE阶段完成初始化。因此，UDS要在DXE期间，以UEFI虚拟设备驱动的形式，运行在UEFI环境中，借助UEFI基础服务完成固件、OS文件检测和异常文件恢复等功能。

1.1 UDS结构

UDS包括4个功能部件：UDSchecker、UDSprocessor、UDSpatcher以及UDSstore。UDS的结构如图1所示。

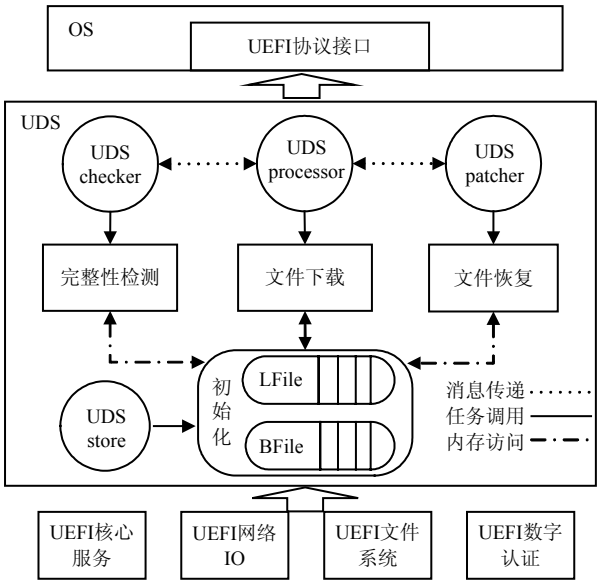

图1 UDS结构

UDSchecker负责检查UEFI模块或操作系统核心文件的完整性：若文件完整，则继续执行；否则，向UDSprcessor发送警报消息alert=(id,url,len)，继续检测。其中id是异常文件名，url是文件的下载地址或文件路径，len是文件的长度。

UDSprocessor收到警报后，首先分配内存addr，并根据alert.url下载文件至addr；然后发送补丁消息patch=(id,addr)到UDSpatcher；最后阻塞执行，等待UDSpatcher回应。UDSpatcher收到补丁消息后，恢复异常文件，解除UDSprocessor阻塞释放内存。

UDSstore负责管理文件加载信息、文件备份信息以及UDS秘钥。文件加载信息在固件或OS文件加载进内存时动态生成；备份信息分成固件、系统文件备份两类，分别保存在硬盘UEFI隐藏分区下的/bak/firmware和/bak/OS目录；这样不仅有效节约了ROM空间，在安装多OS或OS更新时，还能直接对UEFI分区的备份文件进行更新；UDS秘钥用于加密备份文件信息，在开发时硬编码到程序。

1.2 UDS启动

当UDS启动时，UDSstore会创建表LFILE和BFILE；表LFILE中保存加载到内存中的固件驱动或OS文件信息，其表项形式化表示为：(id,start_addr, len)，id是文件名，作为唯一标志，start_addr是文件加载地址，len是内存中文件的长度；BFILE表项形式化表示为：(id,url,fhash)，id是文件标志与LFILE.id相同，url是文件路径或下载路径，fhash是备份文件的数字摘要，用于文件的完整性检测。

在DXE初期，由于操作系统没有启动，BFILE只能保存文件/bak/firmware中的固件文件备份信息。在DXE后期，EFIchecker将对UEFI固件文件进行检测，UDSprocessor以及UDSpatcher则会根据检测结果，实现对异常文件的恢复。

BSD阶段，UDS根据用户选择的OS启动项，对UEFI全局变量OSid赋相应的值；然后读取备份文件/bak/OS/$(OSid)的内容，并将文件信息保存到表BFILE。随后，UDS就可以对OS的核心文件进行完整性检测和恢复。

根据SPEC[1]，UDS还可以注册UEFI系统服务，供其他UEFI程序或OS引导程序调用[5]。注册UEFI系统服务，要封装调用的服务接口，并调用UEFI的Update Capsule机制[1]来实现和OS的信息交互。UDS的启动过程如图2所示。

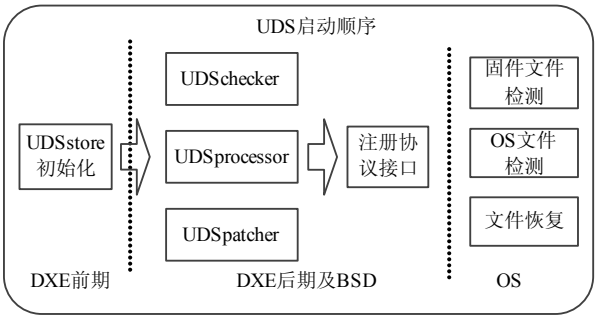

图2 UDS启动流程

2　UDS自我保护

2.1　驱动模块保护

在UDS正常运行的情况下，一旦核心文件完整性被破坏，UDS就能完成检测。如果基于固件的BOOTKIT能够早于UDS获得执行权，就有可能绕过UDS的检测，对计算机系统造成威胁。

通过实验发现，目前大部分固件BOOTKIT的攻击目标主要是OS，通常不会特别地对某个UFEI驱动进行绕过。存在部分固件BOOTKIT，利用固件代码特征或者驱动名称，来定位攻击目标并完成篡改代码、隐藏驱动的目的；在UDS系统的开发和编译过程中，可以对核心代码进行混淆或者修改模块文件名来避免被BOOTKIT定位，从而防范此类攻击。

2.2　备份文件保护

恢复异常文件之前，UDS会根据alert.url到指定的位置下载备份文件。如果攻击者能篡改alert.url地址下的文件内容，或者直接修改alert.url，并将其指向预定的恶意文件，那么当恶意文件被下载到本地加载运行后，就可能造成隐患。因此保证备份文件的安全性十分必要。

为了防止攻击者直接获取备份文件中的重要内容，第三方在开发过程中，可以自定义加密算法，对存在备份文件中的内容进行加密。为了保证下载所得文件的合法性，可以将下载路径设置为操作系统官方网址或系统盘根目录，这样攻击者需要获取官方数据库权限或本地管理员权限，才可能篡改下载路径下的备份文件内容。

3　UDS功能

UDS主要对固件文件、操作系统文件进行完整性检测；并根据检测结果，对异常文件进行恢复。在开发过程中，UDS借助了UEFI的文件系统、网络访问和数字签名等基础服务来实现相应功能。

3.1　备份文件加、解密

UDS启动时，要对表LFile与表BFile进行初试化，将加载到内存中的固件驱动、OS文件信息以及UFEI分区上备份文件的信息保存到相应的表中。为了避免攻击者直接获取备份文件中的重要内容，备份文件将被加密保存；因此在初始化时，要使用硬编码在程序中的秘钥key对备份文件进行解密。

算法1：备份文件加、解密

```
//加密过程
fuction encode(fe, key):
    klen = length(key)
    for (i = 0; i < fe.length; i++)
        //将文件的每位于key相应位异或
        result[i] = fe[i] ^ key[i % klen]
return result
//解密过程
function decode( fd, key ):
    for(i = 0;i < fe.length; i++)
        result'[i] = fe[i] ^ key[i % klen]
return result'
```

3.2　完整性检测

UDSchecke首先利用LFile表项中的文件加载信息计算文件摘要hash；再根据LFile.id获取在BFile中对应的备份文件信息，并将hash与BFile.fhash进行对比：若两个摘要相等，表明文件完整，则继续对下个文件进行完整性检测；否则，表明文件完整性被破坏，则向UDSprocessor发送警告消息alert。

算法2：完整性检测

```
//当表非空时，获取对应表项中的信息
if LFile[i] is not NULL:
        addr = LFILE[i].start_addr
        len = LFILE[i].len
        index = LFILE[i].id
//根据表项信息，计算对应文件的哈希
hash=HASH(mem(addr, len))
//通过标志id关联备份文件，并对比哈希
if hash != BFILE[idx].fhash:
//若不相等，发送alert给UDSprocessor
        send alert = (index, BFILE[j].url, len)
//继续访问下一个表项
```

3.3　文件恢复

当UDSprocessor收到alert消息后：首先分配一片大小为alert.len的内存空间addr，并从alert.url将文件下载到内存addr处；随后UDSprocessor将创建消息patch，通知UDSpatcher对异常文件进行恢复；最后阻塞，等待UDSpatcher回复消息，并释放内存。

算法3：UDSprocessor处理过程

```
while receive alert:
        //分配内存空间
        addr = allocate(len)
        //从alert.url下载备份文件到addr
        copy(download(alert.url), addr)
        //发送patch消息给UDSpatcher
        send phtch = (alert.id, addr)
```

```
//阻塞等待UDSpatcher消息
wait( )
//释放内存
free(addr)
//继续处理下一个alert
```

当UDSpatcher收到patch后，开始对异常文件进行恢复：首先，通过patch.id访问LFile中对应的表项，得到异常文件地址LFile.start_addr；然后用addr保存的合法备份对内存中异常文件进行恢复；恢复完成后，向UDSprocessor返回确认消息，解除阻塞并继续进行处理。

算法4：UDSpatcher处理过程

```
while receive patch
//通过patch.id访问对应的LFile表项
assert(patch.id == LFile[idx].id)
//恢复异常文件
repair(LFile[idx], patch)
//给UDSprocessor返回确认消息
send ACK
//继续处理下一个patch
```

4 实验分析

实验目的是检测UDS的防范能力、空间开销和启动效率。首先，在UEFI开源代码的基础上，编译带有UDS的UDS-UEFI ROM文件；编译好的ROM文件大小为57 KB，能够充分满足固件内存的限制；最后利用周立功SmartPRO 9800 plus编程器，将该ROM文件刷写到UEFI芯片中。测试基于EDKII开发平台，操作系统为Window 7旗舰版SP1，CPU型号为Inter Core Q8200，主板为华硕H81-D，4 G内存条。

防范能力测试用于测试UDS的完整性检查、文件修复及自我保护能力，实验结果如表1所示。BMW、STONED是基于固件的BOOTKIT，实现篡改固件数据和隐藏驱动的功能；TDL4能在固件初期就开始运行，通过HOOK操作篡改固件代码、绕过驱动，劫持执行流程；Superkit[10]、ADORE-NG[11]、hideme和鬼影等是基于操作系统内核的BOOTKIT，主要实现篡改系统调用表、关键函数指针[13]以及修改控制[12]流程等功能。

实验结果表明，UDS完成了对固件、OS文件的完整性检查和异常文件恢复，能有效防范来自基于固件或OS的BOOTKIT的恶意攻击；通过自我防护，UDS能避免BOOTKIT的影响，保持正常运行。

表1 防范能力测试结果

Rootkit	平台	攻击行为	UDS结果		
			完整检查	文件恢复	自我保护
BMW	固件	篡改固件数据	✓	✓	✓
STONED	固件	篡改固件代码、隐藏驱动	✓	✓	✓
TDL4	固件	篡改控制流、绕过驱动、隐藏进程	✓	✓	✓
ADORE-NG	OS	篡改控制流	✓	✓	✓
Superkit	OS	篡改控制流	✓	无	✓
hindme	OS	篡改OS代码	✓	无	✓
鬼影	OS	篡改调用表、关键函数	✓	✓	✓

表2是UDS与其他同类检测工具的对比结果。实验结果表明，Nickle[14]和SecVisor[8]能够对OS核心文件的完整性进行保护，但是不能对固件文件进行完整性检查；UDS和UEFI恶意代码防范系统(MCDU)[5]都能完成固件文件完整性检查、OS文件完整性检查和文件恢复的功能，但是UDS更加精简，所占空间更少。与UDS相比，由于MCDU采用基于恶意代码特征的扫描方法，除了要完成完整性检查及文件恢复的功能以外，恶意代码查询、定位、匹配等功能还需额外的代码实现；另外，MCDU模块中还保存了大量的恶意代码特征和特征定位信息。而UDS的原理简单，不用实现诸如上述的函数，还能将备份文件等信息保存在硬盘分区上，因而UDS的实现简单，配置方便，占有的ROM空间更少。

表2 与其他同类检测工具对比结果

参数	UDS	Nickle	SecVisor	MCDU
固件文件完整性	✓	×	×	✓
OS文件完整性	✓	✓	✓	✓
文件修复	✓	×	×	✓
文件大小/KB	57	>100	>100	>100

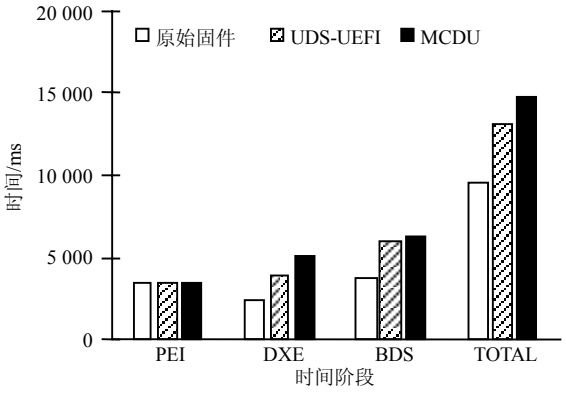

图3 启动时间对比结果

图3是对比UDS-UEFI启动时间的结果。实验表明，UDS-UEFI的启动时间比UEFI恶意代码防范系统(MCDU)要少，不会对计算机启动造成大的影响。由于UDS的检测算法简单，不用实现MCDU中大量的特征识别函数，还能直接利用UEFI平台现成的接口，执行文件读写、数字签名、HASH等操作。因此UDS运行所需的开销较少，UDS-UEFI的启动速度比MCDU的启动更快。

5 结 束 语

本文提出的UEFI固件BOOTKIT检测方法UDS，在操作系统启动之前(DXE阶段)以及OS BOOT阶段，分别对UEFI固件文件和操作系统的核心文件进行完整性检测，并能根据检测结果，对异常文件进行恢复。实验表明，该方法具有启动早、开销少及自我防护性好的优点。

参 考 文 献

[1] UNIFIED EFI FROUM. Unified extensible firmware interface specification version 2.7[EB/OL]. (2017-05-01) [2017-12-10]. http://www.uefi.org/specifications.

[2] WOJTCZUK R, RUTKOWSKA J. Following the white rabbit: Software attacks against Intel VT-d technology [EB/OL]. [2017-12-10]. https://invisiblethinslab.com/resources/2011/Software%20Attacks%20on%20Intel%20VT-d.pdf.

[3] FUTRAL W, GREENE J. Introduction to trust and intel® trusted execution technology[EB/OL]. [2017-12-10]. https://link.springer.com/content/pdf/10.1007%2F978-1-4302-6149-0_1.pdf.

[4] INTEL ADVANCED THREAT RESEARCH. Hacking team's bad bios: Commercial rootkit for uefi firemware [EB/OL]. (2015-10-13) [2017-12-10]. http://www.intelsecurity.com/advanced-threat-research/ht_uefi_rootkit.html_7142015.html.

[5] FU S Y, LIU G S. Research of malicious code defense technology based on UEFI firmware[J]. Computer Engineering, 2012, 38(9): 117-120.

[6] RAO B B, REIJ E A, HALL R L. Trusted support processor authentication of host BIOS/UEFI: US, 20170085383[P]. 2017.

[7] SZEKERES L, PAYER M, WEI L T. Eternal war in memory[J]. IEEE Security & Privacy, 2014, 12(3): 45-53.

[8] KIM, TAESOO. Automatic intrusion recovery with system-wide history[J]. Massachusetts Institute of Technology. 2014, 16(2): 127-135.

[9] POORNACHANDRAN R, ZIMMER V J, SHAHIDZADEH S. Technologies for verifying components: US, 9589155[P]. 2017-03-24.

[10] SUPERKIT INTERNATIONAL COMPANY. Superkit [EB/OL]. (2014-04-19) [2017-12-10]. http://www.superkit.com.

[11] OPENWALL. Adore-ng[EB/OL]. (2014-04-19) [2017-12-10]. http://stealth.openwall.net/rootkits/.

[12] PAYER M, BARRESI A, GROSS T R. Fine-grained control-flow integrity through binary hardening[C]// International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Milan: Springer, 2015: 144-164.

[13] KUZNETSOV V, SZEKERES L, PAYER M. Code-pointer integrity[C]//11th USENIX Symposium on Operating Sys-tems Design and Implementation. Broomfield: USENIX Association Berkeley, 2014: 147-163.

[14] CUI A, COSTELLO M, STOLFO S J. When firmware modifications attack: a case study of embedded exploitation[C]//NDSS. San Diego: The Internet Society, 2013: 159-172.

编 辑 蒋 晓