论文笔记

题目： Digtool: A Virtualization-Based Framework for Detecting Kernel Vulnerabilities

出处： USENIX Security Symposium 2017

作者： Jianfeng Pan, Guanglu Yan, Xiaocao Fan

单位： IceSword Lab, 360 Internet Security Center

原文： https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-pan.pdf

相关材料：
Slide,
Video

一、背景

目前，自动化检测系统内核漏洞的工具比较少，而且，很多的工具都是只能检测开源的系统内核（例如：Linux操作系统），对于不开源的操作系统（例如：Microsoft Windows）则无能为力。因此，很有必要开发一款专用的系统内核漏洞检测工具，在二进制层面检测系统内核潜在的漏洞。漏洞检测工具通常分为两个方面：路径探测和漏洞识别，路径探测通常使用fuzzer工具，查找尽可能多的分支路径；漏洞识别则用于检测这些路径上可能存在的漏洞。而在这篇文章中，作者结合了这两个方面的技术，具体检测了内核中的以下四种漏洞类型：

(1). UNPROBE

在这篇文章中，作者把未经检查的、从用户层传下来的输入缓冲区指针所造成的漏洞叫做UNPROBE。很多内核模块都可能会忽略对用户层的指针进行检查，特别是一些被嵌套的指针，而这种情况是非常危险的，因为它可能会导致非法内存引用、任意的内存读或者写等严重后果。

(2). TOCTTOU (Time-Of-Check-To-Time-Of-Use)

TOCTTOU漏洞来源于对同一个用户层的数据进行多次的访问。在有些系统调用处理例程中，当它要访问某一个用户层数据的时候，它首先会检查这个数据是否合法，然后再使用它，这就会产生两次对该数据的访问，而在这两次访问之间就会存在一个攻击窗口，一旦这个攻击窗口被攻击者利用，就有可能导致非法内存引用、任意的内存读或者写等严重后果。

(3). UAF (Use-After-Free)

UAF漏洞是由于使用了被释放的内存导致的。很多情况下，这种漏洞可能会导致本地提权。在Linux上，像AddressSanitizer这样的工具已经可以用于检测这类漏洞；在Windows上，微软自己开发的DriverVerifier也可以用于检测这种类型的漏洞，但是限制条件比较多。

(4). OOB (Out-Of-Bound access)

OOB漏洞是由于访问了目标内存块之外的内存导致的。该漏洞导致的后果跟UAF漏洞导致的后果一样，都可能会导致本地提权，并且用于检测这两种漏洞的工具也基本一样。

二、提出的方法以及解决的问题

像Windows这样的闭源操作系统，要检测系统中存在的漏洞，就不能再使用基于源码的检测工具，必须使用基于二进制代码的检测工具。因此，为了检测Windows内核中出现的以上四种漏洞类型，作者提出了一个基于二进制代码的检测框架：DigTool。该框架建立在作者自己设计的一个虚拟化监视器（Hypervisor）之上，用来捕获内核执行过程中的各种动态行为，例如：内核对象的分配、内核中的内存访问、线程调度和函数调用等，并基于这些行为，发掘系统中存在的漏洞。

三、技术方法

图1 DigTool的总体架构

DigTool的总体架构如图1所示，它的各个模块分布在系统的不同的层次当中，它们分别包括：Hypervisor中、客户机的内核中以及客户机的用户空间。通过箭头可以知道它们之间的各个模块的相互作用关系：细箭头所连接的两个模块表明它们之间有直接的调用关系或者是直接的传输信息通道，粗箭头表明两个模块之间通过某种事件触发机制进行间接的相互作用。

(1). Hypervisor 组件：

DigTool不依赖于Xen、KVM等当前存在的Hypervisor，而是自己开发一个Hypervisor，它包含三个重要的组成部分：VMM（Virtual Machine Monitor）infrastructure、Interface detection 和 memory detection。

- 虚拟机监控模块（VMM infrastructure）： 首先，它负责检测机器上的硬件环境和操作系统的版本等，以确保DigTool能正确的运行；然后，它再初始化Hypervisor，并把一个原始的操作系统加载到一个VM虚拟机当中运行。

- 接口检测（Interface detection）： 它负责监控用户层的应用程序在调用系统调用的时候传递到内核中的参数。在内核中，它跟踪这些参数被使用和被检查所发生的位置，以发掘潜在的漏洞。为了提高性能，DigTool并不是监控所有的系统调用，它只监控自己感兴趣的系统调用（通过配置文件进行配置）。因此，它限制了被监控的系统调用的范围，以检查特定的系统调用所产生的漏洞。

- 内存检测（Memory detection）： 该模块通过使用SPT（Shadow Page Table）技术，监控客户机操作系统内核中的非法内存访问。为了检测特定的内核模块（例如Win32k等），该模块可以通过调用相应的服务接口（Hypervisor提供给客户机操作系统内核的接口）来限制被监控的内核模块和地址范围。

(2). Kernel-Space 组件：

DigTool在客户机操作系统内核中的所有模块被统称为：Middleware，它属于一个中间桥梁（或者说是一个中间件），用于连接Hypervisor和客户机用户空间程序，具有承上启下的作用。

- 例如图1中，在Loader加载Fuzzer之前，可以通过用户空间中的configuration文件配置被检测的系统调用的范围，然后通过Middleware把相应的信息传送到Hypervisor，这就使得Hypervisor可以在Fuzzer进程空间中检测相应系统调用中的漏洞了。

- 对于接口检测，Middleware通过一个工作线程（Work thread）把所有的相关信息（包括系统调用号、事件类型、事件发生的时间、指令地址以及访问的内存）都记录到日志文件中。以便于日志分析模块（Log analyzer）可以通过日志文件分析并查找出相应的漏洞。

- 对于内存检测，Middleware通过Hook特定的内存操作函数，辅助内存检测模块矫正被检测的内存范围（因为DigTool只检测部分相关的内存范围），并且通过调用Hypervisor提供的接口来限制被监控的内存区域，以提高性能。另外，如果在这个过程中发现一个潜在的漏洞，Middleware还会中断客户机，使之进入单步调试模式，等待外部调试工具（例如 WinDbg）连接，并获取有用的上下文环境，辅助漏洞分析。

(3). User-Space 组件：

为了提高系统的稳定性和健壮性，作者把DigTool的一部分作用功能模块放到用户空间中，这些模块包括：Loader模块、Fuzzer模块和Log Analyzer模块。

- Loader模块，它负责装载一个特定的进程，给DigTool提供一个检测漏洞的进程环境。该模块还需要配置configuration文件，限制被检测的系统调用的范围等（要检测哪些系统调用就把这些系统调用添加到配置文件中）。

- Fuzzer模块，该模块由Loader模块装载，通过Fuzzer模块调用系统调用，并通过调整系统调用的参数，探索尽可能多的路径，使得漏洞检测模块可以发掘尽可能多的漏洞。

- Log Analyzer模块，该模块负责分析日志文件，分析代码中可能存在的漏洞。

四、实验评估

(1). 有效性评估

作者通过测试不同的软件产品来评估DigTool的有效性，这些产品包括Windows操作系统和一些反病毒软件，并且这些产品都是当时最新的版本。实验环境包括Windows 7和Windows 10。为了安全起见，在这篇文章中，作者使用的例子是当时发现的0Day漏洞，并且是已经被相应的厂商修复的漏洞。

Detecting Vulnerability via Interface：

如表1所示，对于Avast Free Antivirus v11.2.2262、Dr. Web 11.0、AhnLab v8.0、Norman Security Suite v11.0.0和Spyware Detector v2.0.0.3这五个反病毒软件，被检测出存在UNPROBE漏洞的数量总共为23个。

表1 被检测出的 UNPROBE 漏洞列表

如表2所示，对于同样的五个反病毒软件，被检测出存在TOCTTOU漏洞的数量总共为18个。

表2 被检测出的 TOCTTOU 漏洞列表

Detecting Vulerabiliry via Memory Footprints:

为了检测UAF和OOB漏洞，作者选择32位的Windows 10为实验环境。检测这两类漏洞不再使用日志的形式，而是使用中断客户机系统的形式，即：当发现可能存在的漏洞的时候，中断客户机系统，等待调试工具连接，等调试工具连接上来之后，可以获取当前产生漏洞的位置对应的上下文环境，用于分析漏洞。但是这种方式的不足之处在于：它需要手工去分析漏洞。

在win32kfull.sys文件中，DigTool最先发现了MS16-123/CVE-2016-7211 CVE漏洞。

而对于OOB漏洞，作者使用DigTool发现了三个win32kbase中的CVE，它们分别是：MS16-090/CVE-2016-3252、MS16-034/CVE-2016-0096 和 MS16-151/CVE-2016-7260。

(2). 效率评估

目前，由于Bochspwn（一个基于Bochs模拟器的内核漏洞检测工具）只能检测前文中提到的四种漏洞类型的一类：TOCTTOU，因此，作者只能将DigTool与Bochspwn进行TOCTTOU漏洞测试比较。基于相同的环境下（相同的硬件、相同的操作系统版本、相同的系统调用参数和相同的测试程序），作者选用了十个最经常使用、最经常被反病毒软件Hook的系统调用的系统调用来测试性能开销。此外，为了测试结果的完整性，作者还选用了一个常用的软件WinRAR来加入这个测试实验当中，这两个工具的性能比较结果如图2所示。

图2 DigTool与Bochspwn的性能比较

在上图的实验结果中，作者分两种情况进行比较:“Unrecorded”和“Recorded”,具体如下：

Unrecorded

在这种测试模式下，在DigTool的配置文件中不添加任何的系统调用（相当于没有内存页面被监控），也不做任何的日志操作，但是在接口检测（Interface Detection）模块中的其它子功能是处于工作状态当中的。由于在检测TOCTTOU漏洞的时候，大部分系统调用和线程都处于不被监控状态，因此，该模式可以反映整个系统的基本性能开销，很适合与Bochspwn进行比较。

该模式下的性能开销比Windows模式（既没有DigTool，也没有Bochs）下慢2.18到5.03倍，比Bochspwn（操作系统运行在Bochs中）快45.4到156.5倍。对于WinRAR的测试结果，DigTool比Windows模式慢2.25倍，比Bochspwn模式快428.8倍。

Recorded

在这种测试模式下，相应的系统调用将被写入到配置文件中，并且相应的行为也会被记录下来（监控配置文件中指定的系统调用和相关的线程，对于无关的系统调用和线程都不做任何操作）。

该模式下的性能开销比Windows模式（既没有DigTool，也没有Bochs）下慢70到90倍，但任然比Bochspwn（操作系统运行在Bochs中）稍快。对于WinRAR的测试结果，这次测试使用极端情况，即：监控NT kernel中的所有系统调用。测试结果表明：DigTool比Windows模式慢13.45倍，比Bochspwn模式快71.8倍。

五、优缺点

优点：

- DigTool相对于Bochspwn来说，在性能方面有很大的优势。不管是在“Unrecorded”模式还是“Recorded”模式下，DigTool都比Bochspwn快。
- 相对于DriverVerifier（一款微软开发的、用于检测Windows内核漏洞的工具），DigTool具有更好的弹性。在检测到一个可能的漏洞的时候，DriverVerifier会造成系统蓝屏死机（BSOD）,而且当这个潜在的漏洞在没有被修复之前，DriverVerifier就不能继续往下运行，因此无法同时检测多个漏洞；而DigTool则不会出现这种情况，DigTool在检测到潜在的漏洞的时候，只会记录漏洞发生的地点等一些有用的信息，不会造成系统蓝屏死机，并且可以同时检测多个漏洞。
- 对于OOB漏洞和UAF漏洞，DigTool在检测到潜在的漏洞的时候，可以直接中断客户机，等待外部调试器（如WinDbg等）连接，查看产生漏洞时刻的上下文，方便分析人员获取有用的信息；而DriverVerifier在检测到潜在的漏洞的时候，只能Crash客户机，并且无法获取产生漏洞时的上下文环境，若要分析产生漏洞的原因，分析人员还需要做额外的逆向分析工作，以确定漏洞产生的具体位置以及具体原因。
- DigTool可以检测UNPROBE和TOCTTOU漏洞类型，而DriverVerifier则不可以。
- 在检测UAF和OOB漏洞的时候，如果漏洞没有造成系统崩溃，则DriverVerifier无法检测到它的存在。
- DigTool发现了45个0Day内核漏洞。
-

缺点：

- 虽然DigTool的性能相对于Bochspwn好很多，但是，由于DigTool需要从Hypervisor和客户机之间频繁的进行切换，性能开销依然比较大。
- DigTool的可移植性依然不是那么好，它目前只能运行Windows平台上。虽然DigTool的Hypervisor层与平台无关，但是它的Middleware层是与特定的平台相关的，要想使得DigTool在别的平台上能够运行，则必须修改它的Middleware。

- DigTool目前只能检测四种漏洞类型：UNPROBE、TOCTTOU、UAF和OOB。对于其它的漏洞类型（例如：空指针解引用、双重释放等）却无法检测到。
- 该工具不是开源的工具，目前网上没有该工具的源代码。
- 该工具只能应用于内核漏洞检测，无法用于应用层的漏洞检测。

六、个人观点

目前，已经存在很多用于检测漏洞的工具，既有收费的工具，又有免费的工具，不同的工具一般都有不同的用途，并且很多工具都只是检测具体的某一类或者是某几类漏洞。它们的检测方法也各不相同，有些基于源码检测，有些则基于二进制代码检测。而基于源码检测的工具比较多，这些工具大部分都是用于检测应用层的应用程序，以及开源的系统内核（例如Linux内核等）。很少有检测Windows内核漏洞的工具，因为Windows内核是闭源的，无法获取到它的源代码。

对于检测Windows内核漏洞，目前有一款功能非常受限的、微软开发的检测工具DriverVerifier，前文也提到，该工具的限制条件比较多，检测的漏洞类型也很少，因此，作者自己开发了一款检测Windows内核漏洞的工具：DigTool，作为DriverVerifier的补充工具。

DigTool主要关注两个部分的内容，一个是系统调用入口，当应用程序调用系统调用的时候，可能会由于各种原因，导致非法的、或者是不适当的参数被传入到系统内核中，如果内核代码不能处理好外部传入的参数，攻击者就可能会利用这些入口点，来获得额外的权限，这可能会导致系统内核沦陷。另一种情况是，内核代码如果对内存操作不当，也可能会导致严重的后果（例如，系统可能会出现崩溃、拒绝服务等现象）。

DigTool是基于二进制代码的一个漏洞检测工具，它的总体架构分为三个组成部分（Hypervisor Component、Kernel-Space Component 和 User-Space Component），其中Hypervisor Component是与平台无关的，而另外两个部分是与平台相关的，如果我们能够把它的Kernel-Space Component（其实就是它的Middleware部分）移植到别的系统上（例如MacOS、Linux OS等），那么，我们也可以利用作者的这个工具检测别的系统上的漏洞。虽然DigTool的User-Space Component也是与平台相关的，但是这部分的移植相对于Middleware来说简单的多，因此，DigTool的可移植性主要体现在它的Middleware上。

作者的DigTool目前已经可以检测四种漏洞类型（UNPROBE、TOCTTOU、UAF和OOB），如果稍作改进，可能也会比较容易检测双重释放等类型的漏洞，因为作者的这个工具可以检测到任意的内存地址空间。作者在文章中说道，当检测OOB漏洞的时候，作者使用AVL树来保存已经分配的内存区域，如果一片内存区域已经被释放，则它将会从AVL树中被移除，如果再次释放该内存区域，则DigTool会首先在AVL树中查找该内存区域，若找到，则释放（一般不可能）；若没有找到，则表明已经被释放，或者是没有该内存区域，即：可以发现双重释放类型的漏洞。但是，有一个很大的问题限制了这个想法，那就是：作者的DigTool不开源，我们无法获取到他的源代码，因此，我们也就无法在他的这个工具上做二次开发。