Computer Science and Application 计算机科学与应用, 2022, 12(12), 2896-2909
Published Online December 2022 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2022.1212294

Hans汉斯

基于硬件性能计数器的恶意软件检测技术综述

户彦飞[1,2]，文　雨[2]

[1]中国科学院大学网络空间安全学院，北京
[2]中国科学院信息工程研究所，北京

收稿日期：2022年11月20日；录用日期：2022年12月20日；发布日期：2022年12月28日

摘　要

随着各种任务交由计算机系统或移动设备处理，大量应用软件走进人们的生活，与之而来的是恶意软件越来越多。对此，主流的恶意软件检测技术弊端凸显，基于硬件性能计数器的恶意软件检测以其独特的优势在安全领域越来越广。据此，本文首先介绍了当前恶意软件的组成及攻防趋势，然后讨论了基于硬件性能计数器的恶意软件检测技术的基本模块，并在此基础上对各项技术中的难点问题进行了阐释，接着梳理了基于硬件性能计数器的恶意软件检测技术的研究现状，最后对其未来发展趋势进行了总结和展望。

关键词

恶意软件，硬件性能计数器，恶意软件检测

A Survey on Malware Detection Technology Based on Hardware Performance Counter

Yanfei Hu[1,2], Yu Wen[2]

[1]School of Cyber Security, University of Chinese Academy of Sciences, Beijing
[2]Institue of Information Engineering, Chinese Academy of Sciences, Beijing

Received: Nov. 20th, 2022; accepted: Dec. 20th, 2022; published: Dec. 28th, 2022

Abstract

With all kinds of tasks being handled by computer systems or mobile devices, a large number of applications have entered people's lives, simultaneously with more and more malware. In this regard, the disadvantages of the mainstream malware detection technologies become prominent, and malware detection based on hardware performance counters is becoming more and more popular in the security field with its unique advantages. Therefore, in this paper we first introduce the definition and classification of current attack as well as the defense trend towards current

文章引用: 户彦飞, 文雨. 基于硬件性能计数器的恶意软件检测技术综述[J]. 计算机科学与应用, 2022, 12(12): 2896-2909. DOI: 10.12677/csa.2022.1212294

malware, then discuss the basic modules of malware detection technology based on hardware performance counters, with explaining the difficult problems in various technologies, followed by surveying the research status of malware detection technology based on hardware performance counters, and finally summarize and prospect future development.

Keywords

Malware, Hardware Performance Counter, Malware Detection

1. 引言

随着科技的进步，电脑和手机的普及极大地提升了处理日常事务的效率，人类的工作和娱乐生活也都离不开软件的应用。截至 2021 年 6 月，我国网民总体规模为 10.11 亿，互联网普及率达 71.6%，手机网民规模为 10.07 亿，网民中使用手机上网的比例为 99.6%。软件应用在方便工作和娱乐生活的同时，也引发了许多新的安全问题。攻击者利用系统中存在的漏洞，窃取个人的隐私数据或者植入恶意软件发起攻击，给国家或个人带来巨大的财产损失。据 CybersecurityVentures 预测，到 2021 年全球因网络安全事件导致的损失将高达 6 万亿美元/年[1]。中国互联网络信息中心发布的第 48 次报告显示，17.2%的网民遭遇网络诈骗，22.8%的网民遭遇个人信息泄露[2]。

恶意软件是指攻击者巧妙的利用漏洞并控制程序的运行行为后，部署在计算机系统上的用来达到他们窃取信息的目的的软件。待恶意软件运行后，计算机内部的文件信息或者其它敏感性信息或者被传输出去或者被加密(如，勒索软件)。2 月 12 日，防病毒软件开发商 Malwarebytes 公布了 2020 年恶意软件预估报告，数据显示，平均每台 MAC 受到攻击的次数要远远高于 Windows PC。Malwarebytes 报告显示，2019 年在每个端点检测到的威胁数量上，Mac 高达 11 次，远远高于 WindowsPC 的 5.8 次。而在 2018 年时，这个数字仅为 4.8，短短一年时间就增加了 230%。而同一时期，在 Windows 平台上，检测到的威胁数量仅仅同比增长了 1%。

随着软件数量的爆炸式增长以及日趋严重的报告，自动化、智能化地识别恶意软件是一个势在必行的趋势。开展针对特定恶意软件攻击类型的防御手段较为困难，因为恶意软件种类和形式多种多样，即使是同一种类型地恶意软件，在保持其基本恶意思想的前提下，存在多种攻击软件的版本。不同的恶意软件种类和形式使得针对特定恶意软件的检测方法更多，导致同一系统上的恶意软件检测过程更加复杂。因此，如何及时地发现系统中各种可能的恶意软件，成为计算机安全人员关注的重点。

旨在检测恶意软件并避免其损害的防御技术已被提出。总的来说，有两种。第一种是静态分析方法，例如，基于签名(如，名称等静态属性特征)的检测技术或基于规则扫描的检测方法。他们检测恶意软件的速度快，但效率低。第二种是动态检测方法，如基于动态行为的检测(如，系统调用[3]等信息流跟踪)，动态二进制检测[4]和异常检测[5]。随着攻击者和防御者之间的竞赛越来越激烈，恶意软件也越来越复杂。因此，动态的恶意软件检测变的资源密集型，也就是说，检测过程会消耗更多的资源并带来极大的开销(例如，对于用软件实现信息流跟踪，10 倍的速度下降是很常见的)。对于移动设备来说，这一问题变得更加严重，因为检测的电量消耗约束了检测工作所付出的努力。

与基于控制流图的动态检测方法相比，硬件性能计数器，因其开销较小且具有深刻揭示软件动态行

为的能力[6]，逐渐被用于恶意软件检测。基于硬件性能计数器的检测手段是以硬件性能计数器记录的恶意软件运行时的数据作为特征数据，并以此作为基础来检测恶意软件；或者通过使用机器学习算法来学习这些特征数据的分布，进而达到检测恶意软件的目的。前者通常成为基于签名的检测方法，后者称为基于硬件性能计数器的启发式检测模型。该类检测模型简化了动态检测过程，解决了常规动态检测恶意软件时的开销问题。由于各种类型的 CPU 中均已配备硬件性能计数器，因此，基于硬件性能计数器的恶意软件检测模型可以很容易地部署。

本文在此背景下，旨在对基于硬件性能计数器的恶意软件检测领域的基本模块进行介绍，在此基础上对各项技术中的难点问题进行了阐释，最后梳理了基于硬件性能计数器的恶意软件检测技术的研究现状，并对其未来发展趋势进行了展望。

2. 恶意软件及其防护

本部分我们简单介绍下恶意软件及其防护。

恶意软件及其创建者。恶意软件是由攻击者创建的目标是损害系统安全或者窃取受害者隐私的软件[6]。这些恶意软件最初创建的时候是为了娱乐，但发展到今天其大多数是出于金融收益的动机[7] [8]。有报道称，在美国获取个人信息、信用卡、敏感机器的登录账户等有活跃的地下市场[9]。同样地，也有政府赞助的机构来创建复杂的旨在为间谍活动或破坏活动而攻击特定计算机的恶意软件[10] [11] [12]。恶意软件的传播有很多种方式，比如，诱导用户点击钓鱼邮件中的链接来下载和安装恶意软件，用恶意的 PDF 或文档文件打开邮件附件，浏览已被利用的网页，使用受感染的 U 盘，或通过移动商店下载那些被重新打包后看起来像正常应用程序的非法应用程序。

恶意软件分类。根据所考虑的方面，可以对恶意软件进行若干区分。经典的恶意软件分类是基于恶意软件的通用名称，一般人在不了解恶意软件分析的情况下经常使用。工作[13]展示了三种恶意软件分类：1) 按类型进行的传统分类，2) 按代码行为进行的分类，以及 3) 恶意软件执行的权限分类，三种分类如图 1 所示。具体地，按类型的传统分类的有关介绍如表 1 所示。

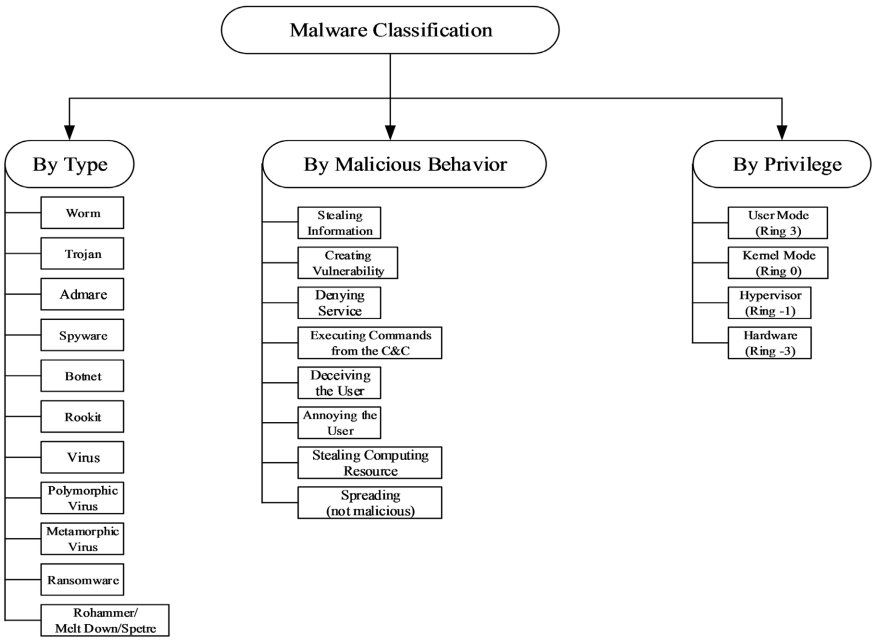

Figure 1. Classification of malware
图 1. 恶意软件分类

Table 1. Malware introduction
表 1. 恶意软件介绍

Malware	Description
Worm	Malware that propagates itself from one infected host to other hosts via exploits in the OS interfaces typically the system-call interface
Trojan	Malware that masquerades as non-malware and acts maliciously once installed (opening backdoors, interfering with system behavior, etc.)
Adware	Malware that forces the user to deal with unwanted advertisements
Spyware	Malware that secretly observes and reports users' computer usage and personal information accessible
Botnet	Malware that employs a user's computer as a member of a network of infected computers controlled by a central malicious agency
Rookit	Malware that hides its existence from other applications and users. Often used to mask the activity of other malicious software
Virus	Malware that attaches itself to running programs and spreads itself through users' interaction with various systems
Polymorphic Virus	A virus that when replicating to attach to a new target, alters its payload to evade detection, i.e., takes on a different shape but performs the same function
Metamorphic Virus	A virus that when replicating to attach to a new target, alters both the payload and functionality, including the framework for generating future changes
Ransomware	Malware that encrypts the user's files (documents and photos) with a strong form of encryption and demands payments in exchange for the decryption key

恶意软件的防护。最常用的针对恶意软件的防护是抗病毒软件(anti-virus software, AV software)。抗病毒软件，不仅可以做到防御病毒，也可以检测和杀掉除病毒之外的恶意软件。典型的 AV 系统在加载文件时，通过扫描文件来查看是否有已知签名，比如恶意软件的代码字符。抗病毒软件的签名生成方式如下：首先，通过蜜罐(Honeypots)采集恶意软件和非恶意软件；其次，将这些软件交由人工分析以生成签名；最后，周期性地将这些签名送到 AV 系统。

在实际中也使用与基于签名的方法成互补的检测方法[14]。在基于声誉的 AV 检测中，用户匿名地向 AV 服务商发送可执行文件的加密签名。AV 服务商然后根据某个可执行文件在很多用户中发生的频率来预测某个可执行文件时恶意软件：通常，以较少数量出现的不常见的可执行文件的签名被认为是恶意软件。这种系统在多态的病毒(Polymorphic viruses)或变异的病毒(Metamorphic viruses)上有效果，但对不可执行的威胁比如恶意的 PDF 或者 DOC 文件等不起作用[15]。此外，它要求用户向 AV 服务商提供他们机器中所安装的程序并且认为 AV 提供商不会泄露这些。

攻防装备竞赛。恶意软件的创建者和检测器之间存在着激烈的竞赛。最早的检测器简单地扫描可执行文件来找已知恶意代码(指令)的字符串。为逃避这些检测，攻击者开始加密有效代码段，作为回应，检测器开始扫描那些与恶意软件打包在一起的解密代码(即不能加密的)。继而，攻击者开始通过使用不同的编译策略(比如选择不同的寄存器赋值或者增加 NOP 操作)随机改变有效代码的主干来生成恶意软件的变体[16]。为应对恶意软件在这方面的变化，防御人员积极寻求恶意软件的行为检测而不再是静态签名。基于行为的检测描述了恶意软件与系统交互的特征：它使用哪些文件，每秒钟指令数 IPC (Instructions Per Clock)，系统调用模式，函数调用以及内存访问变化[17] [18] [19]。使用这些特征，检测器建立正常程序和异常程序行为的模型，并通过与预先建立的行为模式对比来检测异常程序。这些方案的大多数都使用机器学习技术来学习良性和恶意行为，并利用学到的分布来分类[20] [21] [22] [23]。

防病毒软件的效果。和其它任意软件块一样，AV 系统也会有可被利用的缺陷(Bug)。因此，AV 防护也很容易被绕过。Jana 和 Shmatikov 发现他们检查的 36 个商业运行的 AV 系统中，所有的都可以被规避[24]。具体地，他们在解析程序二进制的代码中检测到很多缺陷，那些代码要么允许未经检测的破坏代码通过，要么是需要获取更高的权限。他们认为建立鲁棒性很好的解析器很难，因为文件格式的数量很大，并且大多数的文件格式说明不完整。因此，他们认为试图防御那些复杂的、上百万行的软件代码是徒劳的。和软件检测器不同，基于硬件性能计数器的恶意软件检测器无需处理很多的可执行文件的格式。相反，它们依赖于一种输入格式就行，即来自性能计数器的整数流。而且，它们很难被恶意关闭。因此，基于硬件性能计数器的恶意软件检测器是向鲁棒检测器的重要跨越。

3. 基于硬件性能计数器的检测技术

3.1. 总体构成

基于硬件性能计数器的恶意软件检测技术包括三个模块，即：数据采集模块(Data Collection Module)，模型生成模块(Module Generating Module)，恶意软件检测模块(Malware Detection Module)。其中，数据采集模块负责采集数据；模型生成模块接收数据生成模块输出的数据，并建立模型；恶意软件检测模块以生成模块输出的模型作为检测的基础，对来自数据采集模块的数据做出判断：该采集数据来自良性软件还是恶意软件。接下来，我们依次介绍各个模块及与之相关的技术难点。

3.1.1. 数据采集模块

数据采集模块通过一定的手段记录并读取硬件性能计数器的值。接下来，我们分别讨论数据采集模块的各个模块。

硬件性能计数器。硬件性能计数器(HPC)是性能监控单元(PMU)中的寄存器，主要用于监控程序运行时 CPU 内部架构或微架构上发生的事件。它最初用于系统性能分析和优化[25]、高性能计算[26]或编译调试[27]等。HPC 可监控的事件数量多达 100 个或更多，比如分支加载、分支未命中、缓存引用、缓存未命中、周期数、指令数等，部分 HPC 事件介绍如表 2 所示。表中列出了 HPC 事件的名称、事件的功能介绍以及事件所反映的资源类型，其前 5 行代表体系结构事件，最后 1 行代表微体系结构事件。并且，HPC 在不同供应商或同一供应商的各种类型的 CPU 中以不同数量的方式普遍存在。因此，HPC 可同时监控的事件数量受内部可用的 HPC 的数量限制。

Table 2. Introduction of HPC event
表 2. HPC 事件介绍

事件	功能	记录的资源类型
Instruction Retired	counts the retirement of the last micro-op of the instruction	CPU
Branch Instruction Retired	counts the retirement of the last micro-op of a branch instruction	CPU
Branch Misses Retired	counts the retirement of the last micro-op of a branch instruction and experienced misprediction in the branch prediction hardware	CPU
LLC Reference	counts requests originating from the core that reference a cache line in the last level on-die cache	Cache
LLC Misses	counts each cache miss condition for references to the last level on-die cache	Cache
uops_retired	counts μops that are retired during a clock cycle	Based on definition

HPC 值的读取。根据所运行的操作系统是 windows 还是 Linux，HPC 值的读取方式有所不同。主要思路是在开始执行程序之前，先配置与 HPC 相关的寄存器。一旦程序运行结束或者配置条件满足(如中断发生)，就可以读出相关寄存器的值。对于 windows 操作系统，相关控制寄存器的读写见[28] [29]。此外，也可以通过上层应用直接获取指标，如管理管理器等，但通过上层应用通常获得的是与指标相关的值。对于 Linux 系统，其值的读取可以通过性能监测工具如 OProfile、PAPI 或 Perf 等[30] [31]。

HPC 值的读取方式有两种，既可以在运行程序结束时使用轮询模式读取，也可以在运行过程中使用采样模式定期读取。具体视需求而定。使用轮询模式读取，读到的是一段时间内总的(又称粗粒度的)统计值；而使用采样模式读取，读取到的是细粒度的状态值。目前，采样值的间隔最小可以达到 1ns，采样间隔越小，对程序的运行影响越大；通常选择 10 ms，来折中采样开销和其动态行为的体现。对采样时长的大小，目前并无文献说明应该读取多久，文献[6]提到以 1 分钟的数据作为采集时长，以此生成的模型检测效果较好。需要特别注意的是，部分利用硬件设计漏洞来窃取机密数据的恶意软件(如熔断攻击[32]、幽灵攻击[32] [33])，其运行时间特别短，此时，需要减小采集间隔和采集时长。

获得准确可靠的 HPC 值可能很困难。HPC 值的不确定性来自多个方面。首先，程序每次运行[34]时，其运行环境可能会有所不同。例如，操作系统活动、多任务环境中的程序调度、多处理器交互可能在不同运行之间发生变化。其次，性能计数器可能会对某些处理器上的某些 HPC 事件进行过计数[35]。例如，指令失效事件可能在奔腾 D 处理器上被过度计数。第三，有许多不确定性的来源可能很难预测[35]，比如周期定时器的硬件中断。第四，采集 HPC 值的工具实现方式不同。例如，它们或者从线程级别或者从进程级别监控程序的事件次数，或者在不同的采样时刻读取 HPC 的值，比如中断发生前保存或中断发生后保存 HPC 的值[34]，即使在严格控制的环境中，同一程序运行也可能产生不同的 HPC 值。

通过运行程序来获取 HPC 值时，良性软件可以直接运行，不会对系统安全造成影响，因此，获取良性软件的 HPC 值较为方便。与获取良性软件的 HPC 值不同，获取恶意软件的 HPC 值较为困难。因为，恶意软件并不能像良性软件那样直接运行，否则会给系统带来一定的影响。尤其是在连网的情况下，直接运行恶意软件可能会带来灾难性的损失。对此，通常有两种解决放案：1) 通过运行虚拟机执行恶意软件；2) 通过运行容器。这两种放案都是将所要运行的程序放置于隔离的环境中，以减小对系统的影响。但存在一定的区别，即通过运行虚拟机执行恶意软件，会使 HPC 记录的值不够准确。因为，虚拟机是建立在操作系统之上，并没有提供对主机硬件资源的直接访问。而容器是操作系统级的虚拟化，这种虚拟化与主机操作系统共享相同内核。容器提供了对主机硬件资源的直接访问，不影响 HPC 值的读取[31]。具体地，可以通过先复制基容器，然后在容器内运行恶意软件，当恶意软件无需运行使，直接关闭并毁掉容器。以后再需要容器运行时，可以继续复制容器。使用容器时，HPC 值的读取架构如图 2 所示。

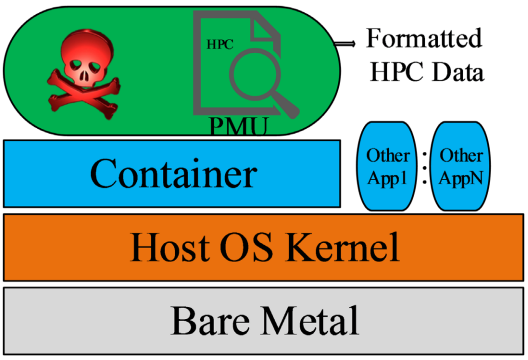

Figure 2. Graph for reading HPC value in a container
图 2. 使用容器读取程序的 HPC 值的示意图

图2中，HPC作为PMU单元中的寄存器，受PMU的单元中的控制寄存器控制。上层监视模块(Monitoring Module)直接与 PMU 单元交互，将读取到的 HPC 值按照一定形式输出。

3.1.2. 模型生成模块

模型生成模块以格式化的 HPC 数据作为输入，通过选定与程序特点相关的 HPC 事件，并结合一定的算法来生成模型。依据 HPC 值的特点，模型生成模块通常使用两种算法来生成模型。对于通过轮询方式得到的程序运行结束后的 HPC 值，模型生成模块通常是利用软件的运行特征，生成基于签名的恶意软件检测模型，比如面向 ROP 攻击的检测。对于通过中断采样得到的程序运行期间的 HPC 的统计值，模型生成模型通常是利用启发式的算法，学习这些 HPC 值的行为分布，从而得到基于启发式算法的恶意软件检测模型。此时，模型算法通常人为设定。具体结构如图 3 所示。

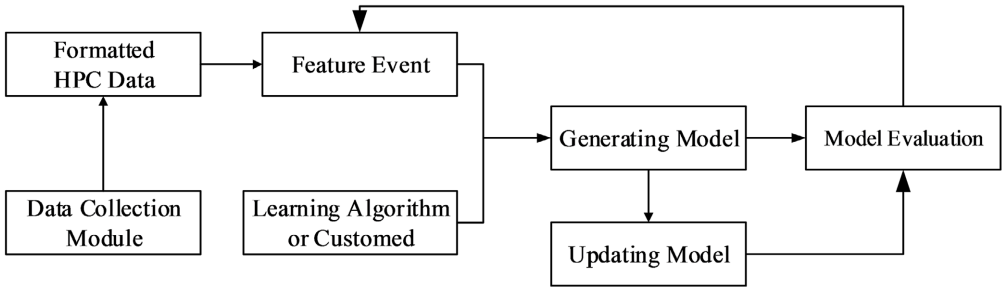

Figure 3. The process of model generating
图 3. 模型生成过程

特征事件选取。模型生成模块的前提是需要选定特征事件，以特征事件的 HPC 值作为模型生成的基础。如前所述，HPC 事件的个数有一百多个甚至更多，因此，首先需要筛选出可作为检测特征的事件，常用的分析方法如计算协方差，其相关计算方法如下。

$$\rho = \frac{\operatorname{cov}(X_i, O)}{\sqrt{\operatorname{var}(X_i)\operatorname{var}(O)}} \tag{1}$$

公式(1)中，ρ 代表皮尔逊相关系数(pearson correlation coefficient)。X_i 是任意 HPC 事件的输入数据集，O 是输出数据集中的不同类别，比如“良性”和“恶意”，i 代表所有 HPC 事件中的任意事件。$\operatorname{cov}(X_i, O)$ 代表输入数据集和输出数据集之间的协方差，$\operatorname{var}(X_i)$ 和 $\operatorname{var}(O)$ 分别代表输入和输出数据集的方差。公式(1)可以进一步详细写成下面所示。

$$\rho(i) = \frac{\sum_{k=1}^{n}\left(x_{k,i} - \bar{x}_i\right)\left(o_k - \bar{o}\right)}{\sqrt{\sum_{k=1}^{n}\left(x_{k,i} - \bar{x}_i\right)^2 \sum_{k=1}^{n}\left(o_k - \bar{o}\right)^2}} \tag{2}$$

公式(2)中，n 是两个数据集中的数据的总数。$x_{k,i}$ 是数据集中对特征 i 来讲的第 k^{th} 个值。o_k 是输出数据集中的第 k^{th} 个值，用此算法可以找到所有(输入)特征事件和输出的相关性关系。相关性的选择阈值可以自行设定，通常认为相关性系数的值超过 0.7 时，已具有较高的相关性。

在此基础上可以进一步细粒度地确定特征事件。具体地，针对基于签名的恶意软件检测模型，通常直接从分析恶意软件的特点出发，选取与特点相关联的事件作为特征；而对于基于启发式算法的检测模型，其确定特征事件的方式是通过后验概率，即先假设是某几类特征，然后通过训练模型并在验证集上测试模型效果，以确定所选取的特征是否符合要求。如果所选取模型效果不好，则继续更换其它特征事件，相反，则认为所选取的特征符合要求。如果采用启发式算法来学习模型，需要对数据进行切分。通

常是先将数据集随机切分为 80%和 20%的比例，其中 80%的部分作为训练集，20%的作为测试集。然后，从 80%的训练集中留出 20%作为验证集，其它用来训练。

模型评价。模型评价的基本思路是将测试集输入到模型中，以观察模型的输出结果。通过对比模型输出结果和已有真值，确定相关的准确率、召回率、误报率、精度、F1-分数、AUC 等。其中，准确率指正确预测的样本占总样本的比例，召回率指正确预测的正样本占总样本的比例，误报率表示被误分到正样本中的负样本占所有负样本的比例，精度表示正确预测的正样本占所有预测为正样本的比例，F-分数是准确率和召回率的调和值，AUC 是横坐标为 FPR，纵坐标为 TPR 所画出的 ROC 曲线下的面积，其值越大，分类效果越好。以上指标的计算方式如表 3，其中，TP 表示将正样本预测为正类，TN 表示将 s 负样本预测为负类，FP 表示将负样本预测为正类，FN 表示将正样本预测为负类。

Table 3. Calculation formula of evaluation index
表 3. 评价指标计算公式

指标	计算公式
准确率	(TP + TN)/(TP + TN + FP + FN)
召回率	TP/(TP + FN)
误报率	FP/(TN + FP)
精度	TP/(TP + FP)
F-分数	$(1+\beta^2) * P * R/\beta^2 (P+R)$

模型更新。模型的更新方式主要有两种。第一种是定期利用新的样本集来更新模型。第二种是通过监控异样样本进行更新。定期更新的方式比较常见，但其缺点也很明显，比如当样本漂流时，不仅是恶意软件的样本发生变化，良性软件的样本随着时间也会发生缓慢的变化，这样会使原有模型不再适用于最新的样本。为克服周期性更新模型所带来的弊端，通过监测漂移样本来实时更新地模型的方式已被提出[36]。此种模型的更新原则是通过将新输入的样本与原模型所基于的样本的分布进行对比，对比的原则则基于低维空间的距离相似性度量。

3.1.3. 恶意软件检测模块

恶意软件检测模块通过利用生成模块输出的模型，以数据采集模块采集到的各程序的 HPC 值作为输入，判断 HPC 值所代表的程序是良性的还是恶意的。

根据对检测时间要求或者对软件漏洞的容忍度，可将恶意软件检测模块以软件实现[31] [34]或硬件实现[6] [37] [38] [39]。软件实现的恶意软件检测模型直接使用生成模块建立的软件模型；硬件实现的恶意软件检测模型对输出模型的软件模型硬线化：即首先通过编译器(比如 Xilinx High-Level Synthesis)，将高级代码转换为硬件语言代码[31]，然后使用工具包实现硬件语言描述。当模型硬件实现时，除了要考虑模型算法的精度外，逻辑实现的面积、能耗和延时也是很重要的因素。一些复杂的算法比如集成学习或者卷积神经网络等可以提升模型的准确率，但同时也增加了硬件实现在逻辑面积及能耗或延时等方面的开销。硬件实现的模型主要有两方面的优点：1) 避免了软件实现时软件自身的漏洞，有更高的安全等级；2) 与软件实现的模型相比，提升了检测时延。

3.2. 基于 HPC 的恶意软件检测研究现状

基于 HPC 的恶意软件检测模型包括大多数监督式的 HMD 模型以及少部分非监督式的检测模型。监

督式的 HMD 模型[31] [34] [40] [41]以 HPC 的状态值为特征，程序样本的标签作为输出，来训练恶意软件检测模型。接下来我们将依据发展脉络穿插介绍这两类模型。

Demme *et al.* 文献[6]验证了 HPC 用于恶意软件检测领域可行性，发现来自 HPC 的数据可被用于检测恶意软件。作者用安卓 ARM 平台以及 Intel Linux 平台上的小部分同族恶意软件的变体作为样本，使用这些样本运行时的细粒度的 HPC 行为数据，结合较为复杂的机器学习方法如 Artificial Neural Network (ANN)以及 K-Nearest Neighbour (KNN)来训练检测模型，验证了基于 HPC 的模型可以检测同族内的更多变体，并且认为这种模型对恶意程序的微小波动具有鲁棒性。进一步地，作者提出可以将此类检测器运行于系统软件层的下面，并对基于 HPC 的检测器的硬件实现做了简单介绍。

尽管 Demme 对基于 HPC 的分类器的硬件实现进行了讨论，但他们没有对硬件实现的开销进行任何分析，特别是能耗，实现面积以及延时也很重要，因为这些指标都决定了硬件实现的分类器的性能。为此，Patel 等人对硬件实现的开销进行了详细分析[31]。为此，作者先使用不同的算法得到离线训练的软件模型，通过详细对比基于 HPC 的分类器的软件实现和硬件实现，Patel *et al.* 得出与基于 HPC 分类器的软件实现不同的结论：即只考虑精度时，准确率较高的软实现模型在使用硬件实现后，其综合性能(包括面积、延时、能耗)反而不够好，而精度不高的模型，其硬件实现后的综合性能很好。

与一级检测模型相比，Ozsoy *et al.*基于计数器的检测模型的软件实现与硬件实现，引入了两级的实时检测模型[38]，并将硬件模型与开源的 X86 可兼容的核集成到一起，运行在 FPGA 上。与基于硬件性能计数器的检测器的硬件实现不同，Ozsoy *et al.*在创建检测模型时引入了其它底层事件，如与内存地址相关的事件。这样，运行于 FPGA 上的检测器可以实时检测，解决了软件运行的漏洞，通过这一级的输出，对恶意软件的优先级进行赋值，然后决定是否使用开销更大的其它基于数据流获控制流的动态检测器。这种做法需要对微处理器的流水线指令做一些微小的变化。

由于恶意软件的精度对上述两级检测模型中的第二级的使用决策比较重要，Khasawneh 提出使用集成学习来提升基于硬件计数器的恶意软件检测的精度。已有的基于硬件性能计数器的检测器都是使用一种算法，由于不同算法对不同的特征的数据的适应度不同，因此，使用单一算法的分类器的性能因使用不同的特征，其精度会存在差别。作者指出将针对特定应用的分类器集成到一起，每个分类器使用不同的特征来训练。需要注意的是，多个分类器应该是用于解决相同的问题，否则集成的意义就会缺失。最后，作者还提出了对分级检测性能的定量评估指标，并将多个分类器使用集成算法后用通过硬件实现，且做到了实时更新。这种使用集成算法的分类器仍具有比软件实现的模型开销小的优点[42]。但作者忽略了一点，即进行集成的多分类器分别使用不同的特征时，需要多个特征。然而，不同类型 CPU 内部 HPC 寄存器通常为 2~8 个，超过 8 个的特征事件需要通过多次运行程序来采集其状态值。而通过多次运行去采样部分特征值，不切实际。对此，Sayadi *et al.* [43]在考虑特征数量的前提下，提出使用集成学习进行高效的实时硬件检测，这种检测综合考虑了单次运行可采集到的特征事件与系统综合性能(准确率、能耗、延时及实现面积)的关系。同时，不像工作[42]使用单一的集成算法和简单的机器学习算法(比如逻辑回归算法)，作者使用了更为常用的 8 种机器学习算法以及 2 中常用的集成学习算法。此外，作者针对工作[38]中需要的对流水线进行改动的情况(因为用到了其它层的语义特征)，也进行了规避。

除监督式的检测器以外，HPC 也被用于非监督式的恶意软件检测。Tang *et al.* [44]以及 Garcia 讨论了使用非监督的方法来检测 ROP 攻击以及栈溢出攻击的可行性。它们以观察到的需要保护的良性程序的 HPC 值作为基础，然后检测 HPC 的波动，以确定是否存在攻击。尽管这种非监督式的方法在检测新的攻击或者攻击者的演变时更有效果，但它们在分析上比较复杂，导致硬件实现比较复杂。同样地，由于计算复杂算法时较大的时延，它们的软件实现也不能很好的实时检测。同样地，文献[45]使用其它非监督的方法，如聚类方法，根据 HPC 测量值的事件序列去检测异常程序。

HPC 不仅可用于检测恶意软件，也可以用于检测某些侧信道攻击，如针对某加密应用发起的缓存侧信道攻击。Zhang [46]使用两个模型来检测云环境中是否存在侧信道攻击。它们使用两个模型，通过相互配合来判断受保护应用的执行，以及与其共同运行的其它虚拟机的运行情况。首先，使用基于签名的检测模型来判断受保护的应用程序的执行情况。当检测到受保护的应用在某个虚拟机开始执行时，立刻采用基于异常的检测模型去检测与该虚拟机位于同一主机设备的其它虚拟机的运行。如果检测到共同运行的虚拟机上出现异常的体系结构事件(如 cache-miss)，则认为有侧信道攻击出现。由于侧信道攻击，尤其是基于 cache 的侧信道攻击，其相关的 HPC 事件反应比较明显，因此，HPC 在检测侧信道攻击方面成为可能。

虽然很多工作提出可以使用 HPC 值以及机器学习模型来检测恶意软件，Zhou *et al.* [47]认为来自 HPC 的微体系结构的信息没有可靠地获取应用程序的本质特征，不能用于检测恶意软件。他们使用了标准的实验步骤和特征选择方法来建立基于 HPC 的恶意软件检测模型。即，先通过主成分分析法确定对预测程序类型有影响的事件，然后建立六种机器学习模型并验证对恶意软件检测的效果，最后得出他们的结论。作者分析排除因实验设置所带来的影响，因此，该结论的影响来源只与 HPC 值的波动有关。此外，结合 Weaver *et al.*对 HPC 值的采样进行的扩展调研[35]，即 HPC 值在各种不同的架构中存在不确定性，且采样波动的数量级在各个架构中是一致的。因此，作者认为在他们所得出的结论在其它架构的环境中同样适用。

即使 HPC 存在于各种类型的处理器中，但常规的用于主机系统或云环境下的恶意软件检测并不适用于低能耗的嵌入式环境中，因为无论时静态检测还是动态检测都需要消耗较大的存储空间，嵌入式设备其存储空间较为紧俏。此外，在嵌入式设备本地进行恶意软件的检测也会额外增加处理器的运算能力。对此，Wang *et al.* [48]提出使用本地采样，远程分析的方式来检测嵌入式环境中的恶意软件。他们通过将采集到的数据进行压缩以减小数据在传输时的带宽，在接收端接收到数据后再进行模型的训练以及恶意软件的识别。这样将数据采集和存储计算分别置于两个环境中，有效地解决了有限计算能力和存储空间的嵌入式平台所面临的问题。

此外，除将 HPC 应用于恶意软件检测的应用研究之外，也有将 HPC 应用于恶意软件检测的理论研究。作者[49]提出了由基于 HPC 的恶意软件检测的分析框架，该框架在仅需要 3 个输入的情况下就可以直接给出恶意软件时良性还是恶意的概率。这三个输入分别时 HPC 的采样率、所需要的一组 HPC 寄存器以及所要监控的程序。整体上，从两个方面进行研究：首先是给定程序的控制流图(control-flow graph, CFG)，另外一个具有相似 HPC 特征的 CFG 能被识别到的概率是多少。其次，研究了恶意软件检测概率随着 HPC 事件及采样间隔的变化规律。最后，指出该理论模型可以扩展到其它检测模型。同样地，HPC 除了应用于建立通用的恶意软件检测器[5]，也可用于进行入侵检测[50]和其它特定恶意软件检测，如固件检测[51]、勒索软件(ransomware)的检测[52]等。

近几年利用硬件漏洞进行的攻击越来越多，如第二部分介绍的熔断和幽灵攻击、Rohammer 攻击等。HPC 除用于检测针对软件漏洞发起的攻击，也可以用于针对利用硬件漏洞发起的攻击，如工作[32] [33] [53]针对熔断幽灵攻击，工作[53] [54]针对 Rohammer 攻击等。这些工作通过观察发起对应攻击时，相关性最高的 HPC 事件，如缓存相关事件或分支指令事件等。

概括来讲，HPC 在嵌入式设备、移动装置、工作站以及云环境下检测恶意软件有着较好的性能，但 HPC 值存在着明显的不确定性[35]。针对不确定性对安全的影响，文献[34]对 HPC 进行恶意软件检测做了系统化的工作：作者首先分析了 HPC 值的不确定性的四个来源，并调研了现有工作在使用 HPC 时对不确定性的考虑；据此，作者指出近十年的工作几乎没有考虑 HPC 值不确定性在安全领域的影响，仅有 4 篇提到了这种不确定性，但他们认为他们的工作不受不确定性的影响。同时，作者还通过对进程进行

过滤，实现上下文切换时 HPC 的保存和恢复，解决了其中一个来源导致的 HPC 的不确定性问题，并讨论了 HPC 过计数时对基于签名的以及启发式的 HPC 检测模型的影响。然而，他们没有研究其它来源(如系统环境中因资源竞争引起的 HPC 波动)对 HMD 模型的影响，也没有提出相应的方法。

近些年来，HPC 已经应用于各种环境中的恶意软件检测，很多工作都使用机器学习算法学到各类检测模型。这些恶意软件检测模型以 HPC 的时间序列值作为输入，输出最终的良性和恶意结果。整个检测过程变得不透明，用户很难理解模型是如何决策的。因此，一些利于安全人员理解的模型被提出来，比如 Pan *et al.* [55]提出基于决策树的恶意软件检测模型，并对决策过程进行解释。同时，为提高模型的检测精度，作者加入了额外的信息(即 ETB)作为特征，并结合 HPC 的特征事件作业检测模型的基础，用于嵌入式设备环境中。尽管文献[55]提出了一个可解释的硬件辅助的恶意软件检测系统，同样地，它仍然忽略系统环境的影响。

上述研究工作都是直接利用 HPC 的行为作为特征来检测恶意软件。这样的检测模型都建立在常用的机器学习算法之上，并且有相似的假设，即样本在测试系统上采集的 HPC 值的分布与训练阶段的值的分布一致。他们忽视了资源竞争对 HPC 级行为的影响。此外，由于资源竞争导致的 HPC 波动是否会对恶意软件检测产生影响，包括[34]在内的这些工作尚未进行研究。文献[56]考虑了运行环境中的资源竞争，证实了竞争环境下启发式模型存在性能下降问题，并提出相应的解决方案。作者以基于启发式的 HMD 模型为研究对象，不考虑基于签名的 HMD 检测模型(比如，使用 HPC 事件值的阈值来检测的 ROP 检测模型[57])，因为签名式地模型很容易失效，不够实用。他们通过设计干扰仿真器来模拟不同资源类型和强度的干扰情况，然后使用深度学习技术找到了竞争环境下程序行为的不变性；最后基于这些不变性的分类器提升了在服务器系统和桌面系统上的恶意软件检测效果。

4. 总结与展望

本文系统概括分析了恶意软件的概念及其分类、当前恶意软件的防御及效果、基于硬件性能计数器的恶意软件检测的技术总体构成及研究现状。现有的基于硬件性能计数器的恶意软件检测技术虽然很好的解决了许多问题，但由于硬件性能计数器的不确定性以及检测模型的黑盒特性，这种检测技术仍然面临许多挑战。我们从以下三个方面分析存在的挑战，并指出未来的研究方向。

1) HPC 不确定性对检测效果的影响。基于工作[34]所作的调研可知，大多数基于 HPC 的恶意软件检测的工作没有考虑 HPC 的不确定性给检测结果带来的影响。由于 HPC 值的不确定性来自于多个方面，因此，需要针对具体方面进行分析和验证。由于系统调度的不确定性以及具有多任务环境的特点，HPC 的采样结果受到系统运行状态的影响，即使是同一程序连续运行两次，HPC 值因运行环境的变化仍会呈现不同的时序值。此外，虽然 Zhou *et al.* [47]给出 HPC 不适用于检测恶意软件，并通过严谨的实验设置排除了因实验设置所导致检测结果较差的原因，也讨论了这种较差的检测结果可能是由 HPC 值的不确定带来的。然而，其没有针对不确定性在恶意软件检测方面进行进一步研究，具体是由哪种不确定性导致的 HPC 值的不准，并对基于 HPC 的恶意软件检测结果带来影响难以确定。因此，对影响 HPC 值不确定性的某一来源进行深入研究，并探索针对这种不确定性的防御方法是有意义的方向。

2) 检测模型的可理解性。基于 HPC 的恶意软件检测模型使用了各种各样的(比如简单的或复杂的)机器学习算法。尽管这些利用启发式算法建立的模型有很好的检测结果(假设有)，但因不同算法其复杂度不同，使得这些模型的决策过程能被安全人员理解的程度不同。对绝大多数模型来讲，在给定输入的前提下，通常是经过一系列黑盒运算，最后直接给出结果。因此，模型的这种黑盒特性使得安全人员对其使用持怀疑态度，尤其是在安全性特别敏感的应用中。因此，对黑盒模型的决策过程进行解释并使安全人员能更好的理解模型的决策过程是重要的研究方向。

3) 检测系统的设计和实现。如前所述，基于硬件性能技术的恶意软件检测由三个模块组成，不同模块对相关领域的技术储备提出了新的要求。恶意软件行为采集模块需要安全人员的分析与处理；模型生成模块需要具有机器学习领域甚至深度学习领域的知识储备；而模型检测模块则是根据批量或流水线的输入直接得到输出，尽管得到的模型可以方便的应用于检测恶意软件，但模型的更新过程，仍需要安全人员进行操作。对此，分模块进行系统设计与实现是未来的一个研究方向。这样既可以使安全人员清楚地认识到相关问题的来源，同时也减少了对机器学习领域技术储备的依赖，更有利于基于 HPC 的恶意软件检测的应用。

基金项目

中国科学院战略重点研究计划(XDC02010300)资助。

参考文献

[1] Ventures, C. https://xueqiu.com/5525633543/174213926

[2] Center, C. I. N. I. http://www.cnnic.net.cn

[3] Cesare, S., Xiang, Y. and Zhou, W. (2013) Control Flow-Based Malware Variant Detection. *IEEE Transactions on Dependable and Secure Computing*, **11**, 307-317. https://doi.org/10.1109/TDSC.2013.40

[4] Dinaburg, A., Royal, P., Sharif, M. and Lee, W. (2008) Ether: Malware Analysis via Hardware Virtualization Extensions. *Proceedings of the* 15*th ACM Conference on Computer and Communications Security*, Alexandria, 27-31 October 2008, 51-62. https://doi.org/10.1145/1455770.1455779

[5] Krishnamurthy, P., Karri, R. and Khorrami, F. (2019) Anomaly Detection in Real-Time Multi-Threaded Processes Using Hardware Performance Counters. *IEEE Transactions on Information Forensics and Security*, **15**, 666-680. https://doi.org/10.1109/TIFS.2019.2923577

[6] Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S. and Stolfo, S. (2013) On the Feasibility of Online Malware Detection with Performance Counters. *ACM SIGARCH Computer Architecture News*, **41**, 559-570. https://doi.org/10.1145/2508148.2485970

[7] Stone-Gross, B., Abman, R., Kemmerer, R.A., Kruegel, C., Steigerwald, D.G. and Vigna, G. (2013) The Underground Economy of Fake Antivirus Software. In: *Economics of Information Security and Privacy III*, Springer, Berlin, 55-78. https://doi.org/10.1007/978-1-4614-1981-5_4

[8] Caballero, J., Grier, C., Kreibich, C. and Paxson, V. (2011) Measuring Pay-per-Install: The Commoditization of Malware Distribution. 20*th USENIX Security Symposium* (*USENIX Security* 11), San Francisco, 8-12 August 2011.

[9] Goncharov, M. (2012) Russian Underground 101. Trend Micro Incorporated Research Paper, 51.

[10] Langner, R. (2011) Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security & Privacy*, **9**, 49-51. https://doi.org/10.1109/MSP.2011.67

[11] Team, S.A., *et al*. (2012) Skywiper: A Complex Malware for Targeted Attacks. Technical Report.

[12] Chien, E., OMurchu, L. and Falliere, N. (2012) W32. Duqu: The Precursor to the Next Stuxnet. 5*th USENIX Workshop on Large-Scale Exploits and Emergent Threats* (*LEET* 12), San Jose, 24 April 2012.

[13] Or-Meir, O., Nissim, N., Elovici, Y. and Rokach, L. (2019) Dynamic Malware Analysis in the Modern Era—A State of the Art Survey. *ACM Computing Surveys* (*CSUR*), **52**, 1-48. https://doi.org/10.1145/3329786

[14] Ramzan, Z., Seshadri, V. and Nachenberg, C. (2009) Reputation-Based Security: An Analysis of Real World Effectiveness. Symantec Corporation.

[15] Bilge, L. and Dumitras, T. (2012) Before We Knew It: An Empirical Study of Zero-Day Attacks in the Real World. *Proceedings of the* 2012 *ACM Conference on Computer and Communications Security*, Raleigh, 16-18 October 2012, 833-844. https://doi.org/10.1145/2382196.2382284

[16] Szor, P. and Ferrie, P. (2001) Hunting for Metamorphic. *Proceedings of the Virus Bulletin Conference* 2001, Prague, 27-28 September 2001, 521-541.

[17] Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M. and Kirda, E. (2010) Accessminer: Using System-Centric Models for Malware Protection. *Proceedings of the* 17*th ACM Conference on Computer and Communications Security*, Chicago, 4-8 October 2010, 399-412. https://doi.org/10.1145/1866307.1866353

[18] Christodorescu, M., Jha, S. and Kruegel, C. (2007) Mining Specifications of Malicious Behavior. *Proceedings of the*

6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Cavtat near Dubrovnik, 3-7 September 2007, 5-14. https://doi.org/10.1145/1287624.1287628

[19] Forrest, S., Hofmeyr, S.A., Somayaji, A. and Longstaff, T.A. (1996) A Sense of Self for Unix Processes. *Proceedings* 1996 *IEEE Symposium on Security and Privacy*, Oakland, 6-8 May 1996, 120-128. https://doi.org/10.1109/SECPRI.1996.502675

[20] Lee, W., Stolfo, S.J. and Mok, K.W. (1999) A Data Mining Framework for Building Intrusion Detection Models. *Proceedings of the* 1999 *IEEE Symposium on Security and Privacy*, Oakland, 9-12 May 1999, 120-132. https://doi.org/10.1007/978-3-540-70542-0_6

[21] Rieck, K., Holz, T., Willems, C., Dussel, P. and Laskov, P. (2008) Learning and Classification of Malware Behavior. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, Berlin, 108-125.

[22] Bailey, M., Oberheide, J. andersen, J., Mao, Z.M., Jahanian, F. and Nazario, J. (2007) Automated Classification and Analysis of Internet Malware. In: *International Workshop on Recent Advances in Intrusion Detection*, Springer, Berlin, 178-197. https://doi.org/10.1007/978-3-540-74320-0_10

[23] Bayer, U., Comparetti, P.M., Hlauschek, C., Kruegel, C. and Kirda, E. (2009) Scalable, Behavior-Based Malware Clustering. *NDSS*, Vol. 9, 8-11.

[24] Jana, S. and Shmatikov, V. (2012) Abusing File Processing in Malware Detectors for Fun and Profit. 2012 *IEEE Symposium on Security and Privacy*, San Francisco, 24-25 May 2012, 80-94. https://doi.org/10.1109/SP.2012.15

[25] Chen, D., Vachharajani, N., Hundt, R., Li, X., Eranian, S., Chen, W. and Zheng, W. (2011) Taming Hardware Event Samples for Precise and Versatile Feedback Directed Optimizations. *IEEE Transactions on Computers*, **62**, 376-389. https://doi.org/10.1109/TC.2011.233

[26] Zhou, X., Lu, K., Wang, X. and Li, X. (2012) Exploiting Parallelism in Deterministic Shared Memory Multiprocessing. *Journal of Parallel and Distributed Computing*, **72**, 716-727. https://doi.org/10.1016/j.jpdc.2012.02.008

[27] O'Callahan, R., Jones, C., Froyd, N., Huey, K., Noll, A. and Partush, N. (2017) Engineering Record and Replay for Deployability. 2017 *USENIX Annual Technical Conference* (*USENIX ATC* 17), Santa Clara, 12-14 July 2017, 377-389.

[28] Intel. Intel® 64 and ia-32 Architectures Software Developer's Manual. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.pdf

[29] AMD (2020) Developer Guides, Manuals & Isa Documents. https://developer.amd.com/resources/developer-guides-manuals

[30] Wang, S., Zhang, W., Wu, H., *et al*. (2015) Approach of Quantifying Virtual Machine Performance Interference Based on Hardware Performance Counter. *Journal of Software*, **6**, 2074-2090. https://doi.org/10.1109/COMPSAC.2015.14

[31] Patel, N., Sasan, A. and Homayoun, H. (2017) Analyzing Hardware Based Malware Detectors. 2017 54*th ACM/EDAC/IEEE Design Automation Conference* (*DAC*), Austin, 18-22 June 2017, 1-6. https://doi.org/10.1145/3061639.3062202

[32] Ahmad, B.A. (2020) Real Time Detection of Spectre and Meltdown Attacks Using Machine Learning.

[33] Li, C. and Gaudiot, J.-L. (2021) Detecting Spectre Attacks Using Hardware Performance Counters. *IEEE Transactions on Computers*, **71**, 1320-1331. https://doi.org/10.1109/TC.2021.3082471

[34] Das, S., Werner, J., Antonakakis, M., Polychronakis, M. and Monrose, F. (2019) Sok: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security. 2019 *IEEE Symposium on Security and Privacy* (*SP*), San Francisco, 19-23 May 2019, 20-38. https://doi.org/10.1109/SP.2019.00021

[35] Weaver, V.M., Terpstra, D. and Moore, S. (2013) Non-Determinism and Overcount on Modern Hardware Performance Counter Implementations. 2013 *IEEE International Symposium on Performance Analysis of Systems and Software* (*ISPASS*), Austin, 21-23 April 2013, 215-224. https://doi.org/10.1109/ISPASS.2013.6557172

[36] Yang, L., Guo, W., Hao, Q., Ciptadi, A., Ahmadzadeh, A., Xing, X. and Wang, G. (2021) CADE: Detecting and Explaining Concept Drift Samples for Security Applications. 30*th USENIX Security Symposium* (*USENIX Security* 21), 2021, 2327-2344.

[37] Khasawneh, K.N., Abu-Ghazaleh, N., Ponomarev, D. and Yu, L. (2017) Rhmd: Evasion-Resilient Hardware Malware Detectors. *Proceedings of the* 50*th Annual IEEE/ACM International Symposium on Microarchitecture*, Cambridge, 14-18 October 2017, 315-327. https://doi.org/10.1145/3123939.3123972

[38] Ozsoy, M., Donovick, C., Gorelik, I., Abu-Ghazaleh, N. and Ponomarev, D. (2015) Malware-Aware Processors: A Framework for Efficient Online Malware Detection. 2015 *IEEE* 21*st International Symposium on High Performance Computer Architecture* (*HPCA*), Burlingame, 7-11 February 2015, 651-661. https://doi.org/10.1109/HPCA.2015.7056070

[39] Ozsoy, M., Khasawneh, K.N., Donovick, C., Gorelik, I., AbuGhazaleh, N. and Ponomarev, D. (2016) Hardware-Based Malware Detection Using Low-Level Architectural Features. *IEEE Transactions on Computers*, **65**, 3332-3344. https://doi.org/10.1109/TC.2016.2540634

[40] Das, S., Chen, B., Chandramohan, M., Liu, Y. and Zhang, W. (2018) Ropsentry: Runtime Defense against Rop Attacks Using Hardware Performance Counters. *Computers & Security*, **73**, 374-388. https://doi.org/10.1016/j.cose.2017.11.011

[41] Singh, B., Evtyushkin, D., Elwell, J., Riley, R. and Cervesato, I. (2017) On the Detection of Kernel-Level Rootkits Using Hardware Performance Counters. *Proceedings of the* 2017 *ACM on Asia Conference on Computer and Communications Security*, Abu Dhabi, 2-6 April 2017, 483-493. https://doi.org/10.1145/3052973.3052999

[42] Khasawneh, K.N., Ozsoy, M., Donovick, C., Abu-Ghazaleh, N. and Ponomarev, D. (2015) Ensemble Learning for Low-Level Hardware-Supported Malware Detection. In: *International Symposium on Recent Advances in Intrusion Detection*, Springer, Berlin, 3-25. https://doi.org/10.1007/978-3-319-26362-5_1

[43] Sayadi, H., Patel, N., Pd, S.M., Sasan, A., Rafatirad, S. and Homayoun, H. (2018) Ensemble Learning for Effective Run-Time Hardware-Based Malware Detection: A Comprehensive Analysis and Classification. 2018 55*th ACM/ESDA/IEEE Design Automation Conference* (*DAC*), San Francisco, 24-28 June 2018, 1-6. https://doi.org/10.1145/3195970.3196047

[44] Tang, A., Sethumadhavan, S. and Stolfo, S.J. (2014) Unsupervised Anomaly-Based Malware Detection Using Hardware Features. In: *International Workshop on Recent Advances in Intrusion Detection*, Springer, Berlin, 109-129. https://doi.org/10.1007/978-3-319-11379-1_6

[45] Garcia-Serrano, A. (2015) Anomaly Detection for Malware Identification Using Hardware Performance Counters.

[46] Zhang, T., Zhang, Y. and Lee, R.B. (2016) Cloudradar: A Real-Time Sidechannel Attack Detection System in Clouds. In: *International Symposium on Research in Attacks, Intrusions, and Defenses*, Springer, Berlin, 118-140. https://doi.org/10.1007/978-3-319-45719-2_6

[47] Zhou, B., Gupta, A., Jahanshahi, R., Egele, M. and Joshi, A. (2018) Hardware Performance Counters Can Detect Malware: Myth or Fact? *Proceedings of the* 2018 *on Asia Conference on Computer and Communications Security*, Incheon, 4-8 June 2018, 457-468. https://doi.org/10.1145/3196494.3196515

[48] Wang, X., Chai, S., Isnardi, M., Lim, S. and Karri, R. (2016) Hardware Performance Counter-Based Malware Identification and Detection with Adaptive Compressive Sensing. *ACM Transactions on Architecture and Code Optimization* (*TACO*), **13**, 1-23. https://doi.org/10.1145/2857055

[49] Basu, K., Krishnamurthy, P., Khorrami, F. and Karri, R. (2019) A Theoretical Study of Hardware Performance Counters-Based Malware Detection. *IEEE Transactions on Information Forensics and Security*, **15**, 512-525. https://doi.org/10.1109/TIFS.2019.2924549

[50] Jyothi, V., Wang, X., Addepalli, S.K. and Karri, R. (2016) BRAIN: Behavior Based Adaptive Intrusion Detection in Networks: Using Hardware Performance Counters to Detect DDoS Attacks. 2016 29*th International Conference on VLSI Design and* 2016 15*th International Conference on Embedded Systems* (*VLSID*), *IEEE*, Kolkata, 4-8 January 2016, 587-588. https://doi.org/10.1109/VLSID.2016.115

[51] Wang, X., Konstantinou, C., Maniatakos, M., Karri, R., Lee, S., Robison, P., Stergiou, P. and Kim, S. (2016) Malicious Firmware Detection with Hardware Performance Counters. *IEEE Transactions on Multi-Scale Computing Systems*, **2**, 160-173. https://doi.org/10.1109/TMSCS.2016.2569467

[52] Alam, M., Sinha, S., Bhattacharya, S., Dutta, S., Mukhopadhyay, D. and Chattopadhyay, A. (2020) Rapper: Ransomware Prevention via Performance Counters.

[53] Li, C. and Gaudiot, J.-L. (2019) Detecting Malicious Attacks Exploiting Hardware Vulnerabilities Using Performance Counters. 2019 *IEEE* 43*rd Annual Computer Software and Applications Conference* (*COMPSAC*), Vol. 1, 588-597. https://doi.org/10.1109/COMPSAC.2019.00090

[54] Aweke, Z.B., Yitbarek, S.F., Qiao, R., Das, R., Hicks, M., Oren, Y. and Austin, T. (2016) Anvil: Software-Based Protection against Next-Generation Rowhammer Attacks. *ACM SIGPLAN Notices*, **51**, 743-755. https://doi.org/10.1145/2954679.2872390

[55] Pan, Z., Sheldon, J. and Mishra, P. (2022) Hardware-Assisted Malware Detection and Localization Using Explainable Machine Learning. *IEEE Transactions on Computers*, **71**, 3308-3321. https://doi.org/10.1109/TC.2022.3150573

[56] Hu, Y.F., *et al*. (2022) Care: Enabling Hardware Performance Counter Based Malware Detection Resilient under System Resource Competition. *The* 24*th IEEE International Conferences on High Performance Computing and Communications* (*HPCC*), Chengdu, 18-20 December 2022, 377-385.

[57] Wang, X. and Karri, R. (2015) Reusing Hardware Performance Counters to Detect and Identify Kernel Control-Flow Modifying Rootkits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **35**, 485-498. https://doi.org/10.1109/TCAD.2015.2474374