IET Information Security

Research Article

IET Journals
The Institution of Engineering and Technology

HPCgnature: a hardware-based application-level intrusion detection system

ISSN 1751-8709
Received on 15th December 2017
Revised 27th March 2018
Accepted on 26th April 2018
E-First on 12th July 2018
doi: 10.1049/iet-ifs.2017.0629
www.ietdl.org

Seyyedeh Atefeh Musavi[1] ✉, *Mahmoud Reza Hashemi*[1]

[1]*Department of Computer Engineering, University of Tehran, Tehran, Iran*

✉ *E-mail: amusavi@ut.ac.ir*

Abstract: In the past decade, commodity software applications have been deployed more than ever in almost every domain. Having the ability to differentiate the original trusted application at run-time from its compromised, mimic or trojanised versions would mitigate a broad range of intrusion threats to these applications. This has been addressed by application-level intrusion detection systems, however, such schemes mostly depend on the system software for either monitoring or modelling the application. This is while system software can itself get compromised by kernel-level rootkit attacks. In this study, the authors have proposed a new hardware-based app-IDS, which works independent of the system software of the target system. The proposed method, referred to as *HPCgnature*, includes a new abstraction corresponding to the repetitious functionalities of programs. Such functionalities generate a distinguishing sequence of periods, referred to in this study as the *Operational Periodicity*. The method uses monitoring scheme based on external access to the hardware performance counters of CPUs. Implementing a prototype, they have shown how HPCgnature can detect intrusions in 12 complex interactive desktop applications. Evaluation results indicate this model could differentiate applications with 98% accuracy, and can detect even small run-time code injection attacks by an accuracy of >75%

1 Introduction

In recent years, computer applications are being used at the heart of almost every industrial plant, medical and strategic computing facility, transportation system and so on. At the same time, the economy behind the malware industry [1], as well as cyber-espionage targeted attacks, have encouraged attackers to exploit vulnerabilities of an executable, inject some new code, redirect the control flow or leverage any other technique to infect an application. A malicious program may also mimic some internal [2, 3] or external [4] behaviour of the original application when doing its own mission at the background. An attack can also be designed to misuse the functionalities of a target program which looks legitimate in the system [5].

Host-based intrusion detection systems (HIDSs) may be able to detect some of the mentioned malicious attacks, however, it has been shown that there are situations that the detection requires levels of application awareness [6]. Having an initially trusted binary application as input, application-level intrusion detection systems (app-IDSs) are specific types of HIDSs try to find anomalous behaviours of the specific application. The approach could be used as a complementary method even if the system is protected by preventive approaches such as binary access control [7], authentication [8], or attestation [9, 10].

Current researches in the field of app-IDS leverage system level models of the initial application, such that the modifications can be detected by the IDS. These models are mostly based on application system-calls [11], events [12], control-flow [13], or data-flow [14, 15]. The models have evolved from naive sequences [16] in primary researches to more accurate models by use of complex techniques such as extreme learning machine [17].

On the one hand, the main shortcoming of existing methods is that they need to leverage dynamic application monitoring facilities inside the operating system (OS), use system/application instrumentation [13, 18] or alternatively use virtualisation [15, 19, 20] in order to obtain their required application traces. In other words, they need to trust the system software, which results in a large and complex trusted computing base (TCB). This is while kernel level vulnerabilities, as well as recently advanced rootkits inside the system software [21–24], have made it challenging for the OS/VMM (virtual machine monitor) to remain in the TCB of the system.

On the other hand, the traditional implicit assumption about the trustworthiness of the OS causes the lack of an independent abstract view of the application in existing app-IDS techniques. In other words, existing approaches usually use the system software's abstraction (e.g. OS system-calls or events) when modelling the application. Considering various kernel-level rootkit attacks, such view would not be robust enough.

As a result, we think that the solution is to have a new app-IDS which includes new monitoring location outside the system SW, a new independent application abstraction, and a new intrusion detection scheme based on these new coordinates. In this paper, we have proposed such an app-IDS referred to as *HPCgnature*. By separating the configuration and measurement from the analysis location, HPCgnature gathers the application data directly from the CPU and outsources the analysis to a trusted monitoring machine. Such an idea requires to have an external communication channel which works independently from elements of the software stack inside the target system. The channel can then be used to forward gathered data to a trusted system.

In order to solve the semantic gap of such fine-grained data read from the hardware, the method also proposes a new application abstraction referred to as *operational periodicity (OP)*. The new abstraction models the application by a set of 〈*periodicity*, *amplitude*〉 pairs, with regard to the main and repetitious functionalities, exist in the today's complex applications. OPs are inspired by Our observations which indicate that the behaviour of an application can be modelled by the superposition of such functionalities. Finally, custom OP analysis techniques have been proposed to be applicable for intrusion detection of different groups of applications.

Our method could be considered trusted as it does not trust the system software of the target suspicious system and decreases its TCB size to include only the CPU. The method can also cover different platforms as it works by a single hardware performance counter (HPC) register available in most existing general purpose CPUs. Although the paper will focus on app-IDS, the method would be applicable in multiple fields, including guest application

introspection, membership test [25] and application forensics. Hence the main contributions of the paper are

- Proposing a new app-IDS with no trust assumption to system software. The method includes a new abstraction of software applications referred to as the Operational Periodicity (OP) which fits the suggested hardware monitoring scheme.
- Implementing a prototype of the HPCgnature and evaluating the idea for 12 interactive complex desktop applications.

The remaining of the paper is organised as follows. Section 2 reviews the related works. Section 3 illustrates the proposing app-IDS, which is then implemented as a prototype and evaluated in Section 4. Taking a glance at future works about other probable applicability of the proposed modelling scheme in Section 6, we then discuss some existing issues about HPCgnature in Section 5. Finally, Section 7 concludes the paper.

2 Related works

In what follows we will first review the host-based intrusion detection systems (Section 2.1) and then the usage of hardware-assisted security and HPCs in the security field in Sections 2.3 and 2.2, respectively.

2.1 Host-based intrusion detection systems

The main differences between host-based intrusion detection techniques are in what they monitor, how they process the monitored data, and where they reside. Existing intrusion detection systems may monitor invoked API call sequences [26, 27], kernel events [12], routine call patterns [28], registry access [29], and network behaviour [30]. While there are a small number of works considering new attribute to monitor [31], most of the existing proposals consider mentioned traditional attributes of the application but with more advanced processes (e.g. hidden Markov models) to achieve behavioural models [17, 28].

Another new thread of research shifts the intrusion detection agent from the OS to a hypervisor and use different virtual machine introspection techniques in order to find their required abstractions. The idea which was introduced by Garfinkel *et al.* [19], has been evolved with more advanced properties such as guest OS portability [20, 32], evasion resistance [20, 33]. In all of these mentioned techniques, the required information is obtained with the help of the OS (in-host approaches) or the hypervisor (out-of-the-box approaches). Hence, they cannot be used when there is no such required trust to the system software.

Recently, there has been a trend to use the X86 System Management Mode (SMM) of the CPU in order to monitor the system and thus the application, while being independent of the OS. SMM mode is a CPU mode traditionally used only by the CPU to do the critical initial tasks such as temperature and power management at boot time. The code executes in a separated address space than the OS in the SMRAM [34]. Spectre [35] was one the first papers which used SMI handler as a periodic application monitoring session. A similar idea is presented in MALT [36] in order to use an SMI handler to communicate with a separate debugger machine. However, the implementation requires overwriting the SMI handler on the system BIOS, which is not a trivial task in practice for all platforms [This is because the SMI handler should be overwritten in BIOS, which in turn requires open source BIOS (like Coreboot [37]) to be available for the platform.]. SMI handling also forces some limitations in size of the code [38]. Finally, SMM mode is not a common mode in current CPUs.

Application intrusion detection systems are one sort of host-based IDS which was primarily proposed by Sielken [39] in 1999. App-IDSs are designed to add levels of context awareness to the existing IDSs [40]. The idea has been evolved to propose context-aware semantic intrusion detection systems for web applications [6]. Having the ability to detect intrusion for a specific application, a new research thread for building intrusion-aware application development frameworks [41, 42] has also emerged in recent years.

2.2 Hardware-assisted security

Securing software layers of the platform with hardware – called hardware-assisted security – is one of the existing trends in the computing industry [43]. The ideas include embedding some hardware features on the CPU chip or other third-party chips, which implements a security solution for software ecosystem. Hardware assisted technologies may be employed in either passive or proactive security approaches. These complementary approaches are often used together in an in-depth defensive strategy.

The idea of Trusted execution environments (TEEs) can be seen as the heart of proactive hardware-enhanced security. The idea is to allow a parallel execution of trusted and untrusted codes in two hardware isolated spaces. An interested reader can refer to [44] to review multiple kinds of TEE implementations.

There are other CPU facilities have been used so far for security purposes. Hardware transactional memories have been used by Liu *et al.* [45] for virtual machine introspection. KBouncer [46] uses Last Branch Record register for ROP mitigation.

When talking about intrusion detection schemes, one prefers not to trust any platform element other than the CPU. Hence CPU hardware facilities are the best choices when assuming to have an untrusted system software stack. HPCgnature is a hardware-assisted app-IDS with a small TCB.

2.3 Security usage of HPCs

While HPCs have been often used in the performance analysis literature, there is a new trend for using them in the context of security. One of the first security uses of HPC was for side channel attacks for cryptanalysis [47] and reverse engineering [48]. Another case is [49] in which Vogl and Eckert show that it is possible to use HPCs to monitor applications by their instructions from a VMM. To this end, they have used trapping selected HPC events to the HMM. HPCs haves been also used for malware detection [50, 51]. Existing works use all available HPC events to gather behavioural data about a malware sample as the initial features for training a detection agent. Another field where the HPCs can help the system security is application integrity. Malone *et al.* [52] have used HPCs for integrity checking of an application, for the first time. Instead of using raw values of the counters, they have built a linear model to reflect the relationship between events of an application. Tang *et al.* [53] used HPCs to detect shellcode execution used in the exploitation of vulnerable programs. Using a base-line model and amplifying a small deviation occurred by exploitation, they evaluated exploitation detection in a pdf-reader application.

Finally, HPC has been also used to find kernel level rootkits inside guest VMs. Numchecker [54] uses HPC counting by a trusted VMM to learn the normal counter values for each system call of the guest OS. Then, it is expected that the counters of the modified system call to take an abnormal value.

The aforementioned methods assume trust to the system software. This is not only because they use software to configure/read the counters, but also because they mostly use software events which are not handled completely by the hardware. In addition, the mentioned researches often require multiple events to be counted by the registers, hence they require software to handle the smaller number of registers that exist in current processors.

Another point is the abstraction level which is used in the existing works. We think using HPC values without considering the appropriate abstraction could not handle neither low interaction/idle intervals in an application trace, nor different combinations of interactions the user may have with today's complex application. Thus, we notice that the evaluations were successful only on simple software applications such as Ps, Ls and so on, or are applied only on a single specific application.

3 HPCgnature

In this section, we will introduce our proposed hardware-based app-IDS. First, we present a general overview of the proposed approach in Section 3.1. Then we will discuss the idea behind our new abstraction intuitively in Section 3.2. Finally, Section 3.3 will

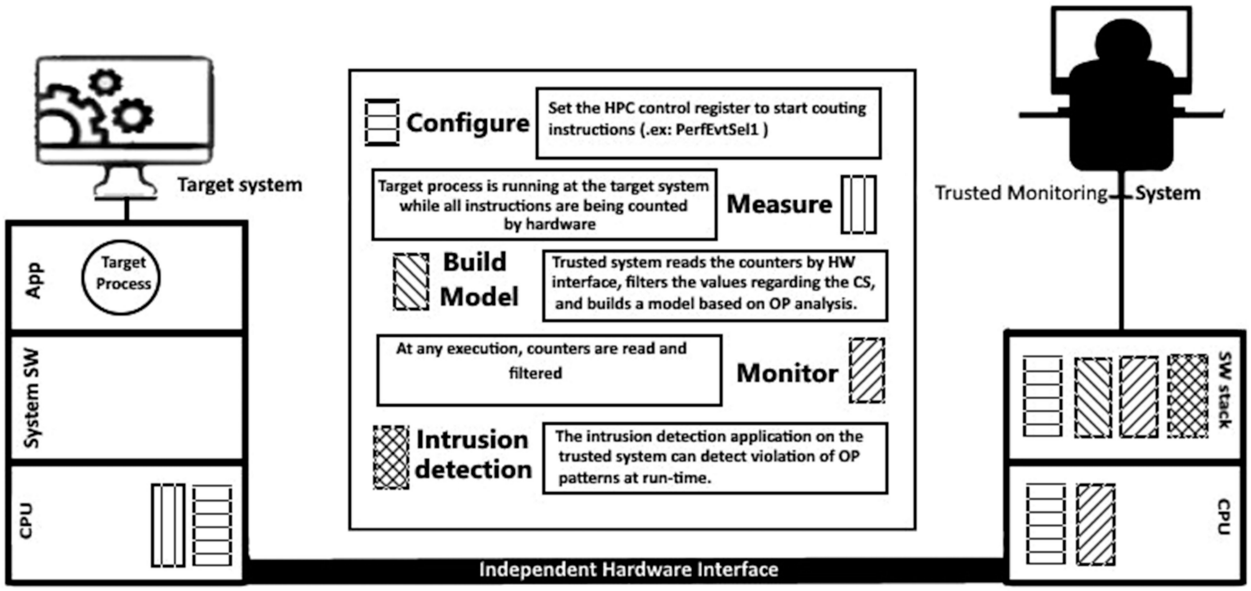

Fig. 1 *Overview of HPCgnature app-IDS*

labelc

1: **procedure** HPCGNATURE
2: Connect to the trusted machine by HW interface
3: Configure the HPC by the interface
4: *train phase*: ▷ Building application model
5: **while** runing process == target process **do**
6: Read HPC values
7: Calulate OP
8: Build application model based on OPs.
9: *App-IDS*: ▷ Intrusion Detection
10: **while** runing process == target process **do**
11: Read HPC values
12: Calulate OP
13: Analyze OP to detect intrusion

Fig. 2 *Algorithm 1 Pseudo code for HPCgnature*

elaborate formally how the abstraction would be applied on applications for intrusion detection purpose.

3.1 Overview

As mentioned above, the objective of this paper is to define a new app-IDS which is independent of system software during all its working phases (including configuration, information gathering and information analysis). However, devoiding use of OS objects (such as threads, events, system-calls etc.) – while keeping the ability to sense what occurs at object level – requires finer-grained information from lower levels of the system. Monitoring at the hardware level would satisfy this requirement. Despite, the occurring semantic gap [55] should be handled when using low level traces. Semantic gap is the challenge of reconstructing abstractions from raw low-level information.

We have addressed this concern by defining a new abstraction on low-level PMU (Performance Monitoring Unit) counters. HPCs are special purpose registers embedded in almost all of the existing CPUs including x86-64, ARM, Cray, Ultrasparc and MIPS processor architectures [47]. The registers are intended to count the occurrence of specified hardware-related activities when enabled. The facility has been used more in performance monitoring for multimedia tuning as well as performance optimisation. Fig. 1 shows the general view of the HPCgnature and the proposed scheme works in practice. We assume to have a predetermined target application running at user-level, on top of some system software components (OS, hypervisor). Since our idea uses only a single CPU register for the target application, the number of protected applications can be few more regarding the number of available physical PMU registers for the specific CPU.

HPCs are configured to count when the application is running. The values are gathered and processed simultaneously by the security analyser in real-time. HPCgnature separates configuration and measurement from the intrusion detection analysis. The former is performed by the CPU of the target system, and the latter is outsourced to a trusted machine dedicated to security monitoring.

We need to use a hardware interface with the ability to configure the CPU. This would be possible using either hardware debug ports or remote management technologies which can access microprocessor at the register level. Many of such devices are currently available for common microprocessors, such as the Intel In-Target Probe (ITP) [56], or the Intel Management Engine Technology [57], to name a few. The trusted monitoring machine is able to configure the CPU of the target suspicious machine using the above mentioned devices. The initial configuring of the counters is performed by setting appropriate values in control registers (like PERFEVTSELx MSRs for the Intel) which are architecture specific [We should note one may decide to read the values by software due to the difficulty of modifying counter values by the OS for evading deep statistical analysis on the application Anyway this is the old trade-off between security and costs which should be handled.].

Counted values are read from each counter register by the hardware interface. Algorithm 1 (see Fig. 2) shows the pseudo code for the steps of the proposed scheme. As the code indicates, the read values are first used to build an application-specific model such that the violation of a model would show an intrusion.

The only point that remains is that the HPCs are counted globally (not per process), thus filtering values for a specific application requires the cooperation of the software to keep counters state at each context-switch. Hence, in addition, to continuously recording counter values, the App-IDS code run on the trusted monitoring machine should also check the page table base register to be aware of context-switches and the currently running process to update its kept states.

3.2 Application's periodicity intuition

Before formally defining our new abstraction, in this section, we are going to give an intuition for the formalism will be proposed in Section 3.3. Every application consists of multiple procedures, initial configuration, library loading, threads, system calls, event notifiers, exception handlers and many other components which run occasionally based on a complex combination of inputs, time and other conditions. Regardless of what a software is intended to do, it can be seen as the superposition of the sporadic run of such components.

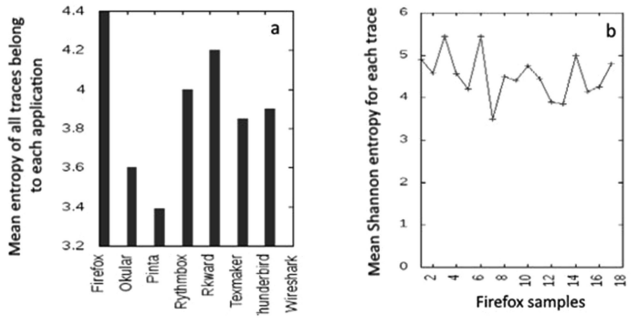

Fig. 3 *Different executable have different inherent periods*
(a) Mean Shannon entropy of periods found inside traces of various application, ***(b)*** Exact Shannon entropy for periods of traces of Firefox application

Table 1 Sample available hardware events in an Intel core-i7 CPU

Events	Speculative	SW-independent	granularity
branch-instructions	Y	Y	branches
branch-misses	Y	Y	branches
bus-cycles	N	Y	memory/IO
cache-misses	N	Y	data in cache
cache-references	N	Y	data in cache
CPU-cycles	N	Y	instructions
ref-cycles	N	Y	instructions
instructions	Y	Y	instructions

Let's think that we know the exact size of each component and the sizes are sufficiently scattered. Hence if we can have an estimation of the size of code run by the application between small time intervals, we can make a reasonable guess about which components have been run at each interval. In other words, we can build a signature for applications based on the components run inside the app. As a simple example imagine a simple system with two system-calls which their handler execution is estimated to take 3 and 22 ms. Two application's system-call execution time shows that programs A and B were running call handlers for 44 and 25 ms, respectively. We can conclude that program A calls the first system call twice, and the program B calls both of existing system calls.

In practice, however, obtaining the accurate size of components and decomposing the application is not a trivial task. Our idea is to leverage mathematical tools which find a set of periodic behaviours embedded inside time-series. To see how much the idea can be successful in practice we have used such a mathematical tool (which will be discussed soon in Section 3.3) to decompose each application execution with a set of ⟨period, amplitude⟩ pairs. Fig. 3*a* shows how much such periods are diversified between nine different complex interactive applications. The diversity is measured by the mean Shannon entropy of the periods belongs to a specific application. As the figure suggests the periods looks to be diversified enough to be used as the building blocks for our app-IDS scheme.

One may ask about how the dependency of each application traces to the input is handled? We think that the size of code run originally by a program at the application level is often dominated by its corresponding low-level codes because of today's programming languages with high abstraction level. In fact, with current high speed CPUs, a process can run up to billions of instruction per second. This is actually a summation of what the application-level code does and the codes invoked in lower levels of the system (interpreters, libraries, and OS) [to translate the original code to native code run on top of the hardware]. As the result, we have observed that although the input of the interactive programs can affect the analysis, however, the effect is not such enormous when compared to the inherent behaviour of the application. This can be seen in Fig. 3*b* which shows the mean entropy of periods found in different run of Firefox browser. Different runs have been tried to cover diverse surfing behaviours. The entropy looks to be converged between different traces. Despite, as we will note in Section 4, sometimes such input dependencies can be filtered by appropriate filters in the frequency domain.

This dependency on inputs, however, is more important for the simpler executables with a lower number of functionalities. To check this intuition, we have studied such kinds of periodic behaviour analysis on ELF binaries exist in the VX-Heaven malware dataset [58]. The set includes about 180,000 malware binaries with minimum and maximum sizes of 1 and 32 KB, respectively. As we expected, malware samples do not show such distinguishable behaviours, as the result of their small mean size.

3.3 Application intrusion detection

In the previous section, we have assumed to have an estimation on the size of running code between small intervals, however, in practice, we have multiple options to achieve such estimation using HPCs. Hence an important decision point is about which event to monitor? on the one hand, it should be fine-grained to be able to distinguish between similar applications; and at the same time, it should be sufficiently abstract to ignore the dispensable details which impose a great performance penalty on application execution, as well as input dependencies.

The chosen event also should be non-speculative and deterministic [59], which means being independent of system load, branch predictor, TLB statistics and so on. Due to the difference in the number of HPCs and also the available hardware events among different microprocessors, the indicator should be chosen from the common events. As an instance, Table 1 represents the available hardware events in an Intel CPU. With the mentioned constraints in mind, counting instructions is an appropriate choice existing in all architectures.

One may argue that branch related features can also be a choice. However, there are two limitations considering such kinds of features. First, an attacker may use injection to execute the whole attack payload in the context of the attacked executable or he may only execute few tricky codes for some mission (e.g. disabling some security protections). Branch-related features are not as effective as instructions in the later scenario for a small amount of code is executed in the context of the protected application. Second, the OP analysis (as will be discussed in Section 5) is concerned by periodicity with measured values (and not only the raw values). Hence it requires to use features with sufficiently considerable values and periodicity. It is worthy to note that one may propose another HPC analysis technique (other than OP analysis) which can benefit from branch-related features.

Hence by use of HPC instruction counter as the application indication, we now can define the application instruction series as the following time series:

Definition 1: The application instruction-series (ins_A) is defined as a sequence of

$$\text{ins}_A = \{(\text{HPC}_{\text{INS}})_n\}$$

In which $(\text{HPC}_{\text{INS}})_n$ corresponds to the value read from HPC instruction counter register at the nth sampling unit of the execution time of the application A.

The sampling rate of instructions can affect the accuracy of the OP analysis. Thus we need to use fine-grained sampling rate. This can be decided based on the CPU and HW interface speeds in practice. After modelling an application as time series, we can extract the corresponding periodic behaviours embodied in the time series. This can be done using spectral density estimation techniques which find dense regions in the frequency domain. As was mentioned before, we think these dense regions in the frequency domain of an application can be mapped to its main functionalities. Thus analysing the frequency/period of such regions can help us to distinguish between applications with no need to know about the detailed functionality.

Hence we define application OP as a sequence of different dominant periodic behaviour which can be extracted from the

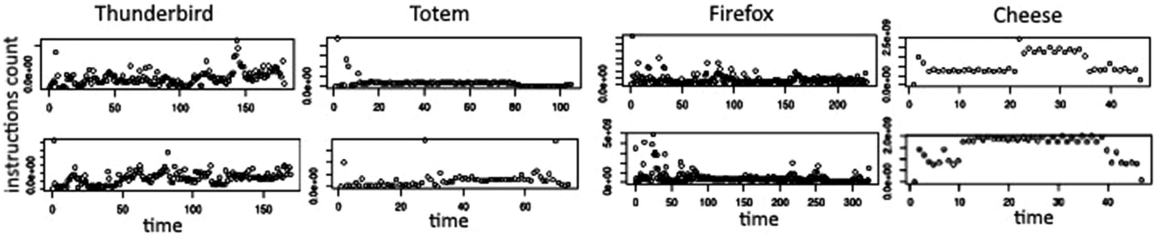

Fig. 4 *Two samples from HPC instruction counters for some applications to show the distinctive counter pattern for each. Applications include Thunderbird email-client, Totem video-player, Firefox browser, and Cheese webcam application*

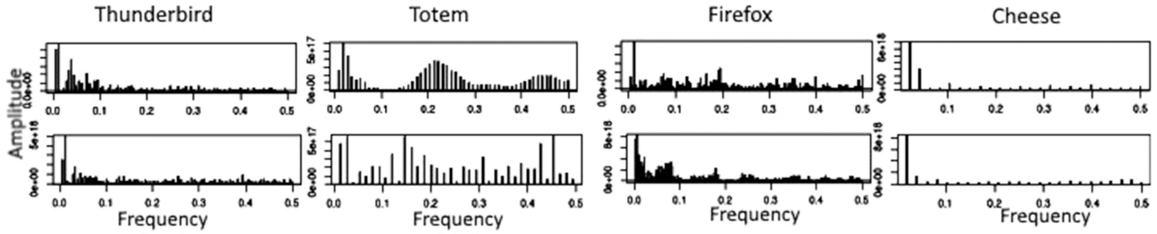

Fig. 5 *Relative OPs calculated for each sample in Fig. 3*

application instruction-series. We can define the OP of an application formally as follows:

Definition 2: The OP of an application instruction-series (ins_A) is defined as a sequence of p_n and e_n:

$$\text{OP}_{\text{ins}_A} = \{(p, e)_n\}$$

In which p_i corresponds to the normalised periods detected in the instruction series A, and e_n is its corresponding estimate for the domain of period p_i.

One of the advantages of using spectral density estimation is that it does not depend on the size of different traces. In fact, since it looks for periodic behaviours in a trace, it can be applied on a small interaction of the user with the program as well as very long traces. It also can wipe out the idle intervals of the application when the user does not interact with the program for a while. In addition, the normalised results also ensure that the OP of an app is a vector containing frequencies in the range of (0; 1]. This would help us to compare different applications with different frequencies. HPCgnature builds a hardware-based model for complex software applications by their OPs. Other than the intrusion detection, this will be applicable when applying white-listing policies, detecting mimic applications, and in application detection for virtual-machine introspection. By obtaining OPs for each application, the behaviour of each specific application can be trained and modelled using several machine-learning techniques. In the next section, we will see such trained model in practice.

4 Evaluation

In this section, we will describe the implementation of HPCgnature prototype (available in [60]).

For evaluating how can HPCgnature be used to distinguish between different applications, we have used a dataset of 12 applications which are chosen from popular desktop interactive applications from different categories (text editor, multi-media, camera etc.). Each application has been run for several times and traces were gathered which include the HPC values for instruction counters, counted at every second. We have logged the traces from the start of an application until the end time. Experiments are done by a laptop having an Intel corei7 CPU and Ubuntu 15.10 OS.

Our proposal requires to use hardware debugging device or alternatively hardware out-of-band control technologies to configure, read, and purify the values. However, as a prototype, we have used Perf library [61] for reading the registers. This does not affect the OP analysis, which should be applied to the hardware read data. Perf includes some useful utilities for configuring and handling per-process read of registers. Regardless of the SW/HW reading scheme, counting mechanism does not affect the program performance as it is a hardware measurement. Since we are reading the values by the software we have observed that choosing finer-grained time units smaller than 1 s, would result in empty returned values which indicates insufficient time for the library to read and record the value. Thus, our implementation is done by counting instructions per second.

We have implemented all computing codes by *R* statistical ELF software for further processing of traces and frequency calculations. In all experiments, traces are gathered from the real interaction of a user with each application, such that different possible inputs and behaviours can be covered. All gathered data are available in [60].

In the first step, we recorded 30 traces for each application. Fig. 4 shows the examples of distinctive patterns of instruction counts in the time domain for some sample applications. The figure verifies our first intuition about the diversity of such patterns for identifying applications. In order to find out the differences in the natural periodicity of applications in the frequency domain, we have applied a Butterworth low-pass filter on the application traces. This would remove noises and purify the inherent frequencies of the application.

In the next step, we should obtain the OP vectors for the gathered traces to achieve better accuracy than by mean entropy measure (was seen intuitively in Section 3.2). Fig. 5 shows some sample OP traces of the applications were seen in Fig. 4. As the figure suggests, the frequency elements paired with their corresponding estimation value have made a specific distinguishing model for each application. Thus we then have processed raw OPs in order to obtain a feature vector we have defined in the form of

$$V_i = (P_1, P_2, \ldots, P_{40}, A_1, A_2, \ldots, A_{40}, P \odot A, P \otimes A)$$

In which P_i and A_i indicate the operational periodicities and their relative amplitude, respectively.

Different traces have a different number of periodicities; thus we have considered first 40 ones which seems to be present for all traces. The next two features include the inner and cross product of periodicity and amplitude vectors.

In order to investigate the effectiveness of the proposed app-IDS we then used these features, to train a classifier using 10-fold cross validation. Using a C5 tree (with the depth of 13 levels), we were able to classify the traces with the accuracy of 98.16% accuracy. The model can be viewed as a decision tree with ten rules. Misclassified samples (1.81%) belong to the Rythmbox and Texmaker applications.

In order to simulate a compromised version of the applications, we have used Linux-inject utility [62] to inject a pre-compiled

library into the application at run time. The tool uses _dl_open function which can be found in all Linux processes to load the desired library. We have used 14 malicious files with.so extension extracted from the malware attacks in the wild, which were downloaded from VirusShare malware repository [63].

Running such injected applications requires an isolated environment in order to gather the traces, thus virtualised HPCs are required. There are multiple approaches to measure the guest's performance counters. We have used VMware Workstation which emulates non-speculative events [59]. Thus our experiments in this section being done on a VMWare guest with Ubuntu 14.10 OS. Here we have gathered >60 traces for each application including both ordinary run and also compromised run and then we have calculated the OPs for each.

Library injection is reflected very imperceptibly and noisy. Thus more processing effort should be done to extract the abnormal behaviour from the OP traces. There is an intuition which can help us find these invisible effects: OP components have not same weights. Lower frequencies which show the inherent frequencies of an application are more important when we are looking for a specific application among many other applications. However, high frequencies can be used in order to identify a compromised sample between some original benign versions of the same application. Despite, we should note that software attacks are not the only source of high frequencies. Some other transient pulses may be due to the high volume of I/O, exceptions, less used program options, mistaken interactions of the user and so on. Thus, after removing low frequencies, we can focus on more realistic candidates for abnormal events. As was mentioned in Section 3.3, further analysis would be application specific, regarding the nature of each application. In continue we will see three examples of such processing:

• Rhythmbox audio-player/Totem video-player: In infected traces the attack point has caused an abnormal change-point in the curve. This may be because of regular sampling methods exist in codec-players, which dominate the frequencies of user activity. Thus by finding the outlier point between such change-points the attack was detected. We have used Mood change point detection algorithm [64] to find the change points.

- Firefox: A browser working frequency is highly dependent on the speed of interactions, and can vary from time to time. Thus the frequency of an injection attack is not simply found. However, choosing a sufficiently high-pass filter to ignore low frequencies, a small peak would be often visible in infected traces.
- Libre-office: A good way to reflect the difference in traces is regarding the AUC (Area Under the Curve) of such high frequencies to whole AUC. This kind of analysis seems more suitable for applications with higher levels of interaction with the user.

Table 2 shows the results for anomaly detection of mentioned applications. It is worthy to note that gathering a wider range of application usage in an organisation, one can use machine-learning methods to extract and learn more common behaviour pattern to improve the results.

Table 2 Application anomaly detection results

Application	Accuracy, %	False positive, %	False negative	Method
Totem video-player	82	14	25	abnormal change point
Rhythmbox audio-player	78	17	6	abnormal change point
Libre-office writer	76	7	3	AUC
Firefox	74	25	29	high-frequency peak

5 Discussion

There are some additional items should be considered for the HPCgnature to be effective which will be discussed in this section. The first point is about the granularity of the analysis. OP analysis could be helpful only if is used in a fine-grained manner (small sampling rate, the accurate process of time series etc.). In Section 4, we have seen that the change in counter values occurs in injection scenarios, would disclose such attacks. Training configuration should be studied more for other kinds of attacks.

Another challenge is how to find a covering trace set of an application. A complex application may behave more diversely if it is considered as a whole entity, however, considering different legitimate paths can result in more accurate analysis. Although the current profiling literature – which is more concerned with the distribution of task response-times – looks coarse-grained for security related attack detection, however, some statistical ideas exist for profiling multi-modal timing behaviour [65] or workload-sensitive [66] software can be mapped for modelling complex software with multiple menu options and behaviours. Another difficulty in mapping noted topics is that the source code is not available when working with commodity software.

Yet another consideration for getting better results is about the target application can be used in OP analysis. OP analysis tries to follow the traces for different frequencies in an application, thus it best-fits for more complex interactive programs. In fact, it would not be suitable for simple command line applications which do not include such frequencies regarding different functionalities.

In addition, there are some sources of noises in an OP trace inherently. The first one is the context-switch noise. the process of keeping the state of HPC values for a specific application between context-switch occurrence will add some noises to the traces [47] which could be considered more accurately regarding a specific machine. Another noise source is due to the long duration of execution, which is the result of more interrupts (time/IO) to the application [67]. Decreasing these set of noises, one may have the ability to identify applications more accurately.

One of the key points in HPCgnature is to use a single pure hardware event which is available on all processors. Although our proof-of-concept was not implemented by a hardware debugging port, however, as the chosen event does not rely on any software process in the target system, it would be possible to be used by a hardware debug equipment. This is while most of the existing proposals rely on other events which are derived from such original hardware events obtained by a software library process.

Finally, it is worthy to note that how difficult is to bypass the HPC analysis. Maintaining complex statistical measures based on HPC values is not a trivial task for an attacker. Hence we think that an ideal HPC analysis method could find many types of intrusions. However, the power of an HPC analysis technique relies on the granularity of its statistical analysis technique. Hence, there is no theoretical limitation for app-intrusion detection. Our prototype, however, may have not such an ideal analysis power.

6 Future works

Although HPCgnature was proposed to be used for intrusion detection, however, there are multiple other problems that the scheme can be applicable there also. In fact, wherever the problem can be viewed as an identification problem, the hardware-based modelling scheme embodied in HPCgnature can be a choice. As an example, we have tried to show how can OP traces disclose if the application is running inside a virtual environment. Thus we have provided same experiments on applications once on a physical machine and then on a virtual machine. Fig. 6 shows the result of such same experiments on the Pinta GNOME image software and Rhythmbox audio editor. We have measured the maximum domain of periods with lower frequencies, which has been obtained by applying a Butterworth low-pass filter on the application's OP trace. As was discussed in Section 4, these low frequencies can indicate inherent behaviours of an application.

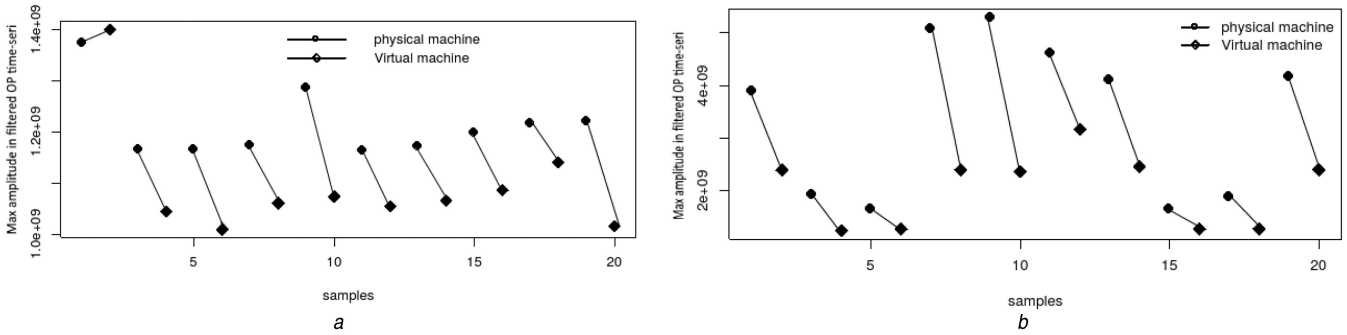

Fig. 6 *Differences between similar application activity in a physical and virtual machine. The maximum amplitude of OP time-series is shown*
(a) Traces for Pinta application, ***(b)*** Traces for Rhythmbox application

As the figure suggests a measured value for physical machines is bigger in all experiments. Although this decrease in measured value is evident per experiments in both applications, we will see that there are Rhythmbox experiments in which low values for either of environments has been observed. These are experiments where the initial music import phase occupied more considerable time than the typical audio-playing operation of the application.

Anyway, in practice, one would use the mean measure of a bulk operations of an application to a judge about its environment which would reflect such decrease evidently. Despite this raw idea should enrich to cover the time cheating scenarios shown to be possible when having a malicious hypervisor. Another future work thread can be use of better statistics scheme to improve the accuracy.

7 Conclusion

In this paper, we have proposed the idea of using HPC registers in the context of an application intrusion detection system. While the existing researches in the literature depend on the OS as a trusted system component for gaining a detailed view of an application, we have proposed an app-IDS which acts independently both for monitoring and modelling phases. This independence to the untrusted software stack of the target system is achieved by the cost of preparing a secure independent communication channel to a trusted monitor system. We believe providing such a channel is worth spending the required extra cost for the hardware interface.

Our HPCgnature scheme uses hardware components for external access to the hardware for monitoring, and frequency-based abstraction of programs, referred to as operational periodicity, for modelling. This can be worthy as it needs only to a single purely hardware counter, which exist in almost all commodity processors. In other words, our scheme does not need to some events which are counted by the OS breakpoints or similar available events require untrusted software exist in the target system. We have implemented a prototype of the proposed scheme and shown how can it be helpful in practice.

8 References

[1] Gutmann, P.: 'The commercial malware industry'. DEFCON Conf., Las Vegas, USA, 2007

[2] Kruegel, C., Kirda, E., Mutz, D., *et al.*: 'Automating mimicry attacks using static binary analysis'. Proc. of the 14th Conf. on USENIX Security Symp., Anaheim, CA, USA, 2005, vol. 14, pp. 11–11

[3] Parampalli, C., Sekar, R., Johnson, R.: 'A practical mimicry attack against powerful system-call monitors'. Proc. of the 2008 ACM Symp. on Information, Computer and Communications Security, Tokyo, Japan, 2008, pp. 156–167

[4] 'Newly found malware can steal bank details on android phones', 2016, Available at https://www.hackread.com/malware-can-steal-bank-details-android-phones/

[5] Langner, R.: 'To kill a centrifuge: a technical analysis of what stuxnet's creators tried to achieve', 2013, Available at http://www. langner.com/en/wp-content/uploads/2013/11/To-kill-a-centrifuge.pdf

[6] Viljanen, L.: 'A survey on application level intrusion detection', 2005, Available on: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.297.8917&rep=rep1&type=pdf

[7] Wu, Y., Yap, R.H.: 'Simple and practical integrity models for binaries and files'. IFIP Int. Conf. on Trust Management, Darmstadt, Germany, 2015, pp. 30–46

[8] Halim, F., Ramnath, R., Wu, Y., *et al.*: 'A lightweight binary authentication system for windows'. IFIP Int. Conf. on Trust Management, Trondheim, Norway, 2008, pp. 295–310

[9] Coker, G., Guttman, J., Loscocco, P., *et al.*: 'Principles of remote attestation', *Int. J. Inf. Secur.*, 2011, **10**, (2), pp. 63–81

[10] Gu, L., Ding, X., Deng, R.H., *et al.*: 'Remote attestation on program execution'. Proc. of the 3rd ACM Workshop on Scalable Trusted Computing, Alexandria, VA, USA, 2008, pp. 11–20

[11] Hu, J., Yu, X., Qiu, D., *et al.*: 'A simple and efficient hidden Markov model scheme for host-based anomaly intrusion detection', *IEEE Netw.*, 2009, **23**, (1), pp. 42–47

[12] Rhee, J., Zhang, H., Arora, N., *et al.*: 'Software system performance debugging with kernel events feature guidance'. 2014 IEEE Network Operations and Management Symp. (NOMS), Krakow, Poland, 2014, pp. 1–5

[13] Basu, S., Uppuluri, P.: 'Proxi-annotated control flow graphs: deterministic context-sensitive monitoring for intrusion detection'. Int. Conf. on Distributed Computing and Internet Technology, Bhubaneswar, India, 2004, pp. 353–362

[14] Castro, M., Costa, M., Harris, T.: 'Securing software by enforcing data-flow integrity'. Proc. of the 7th Symp. on Operating Systems Design and Implementation, Seattle, Washington, 2006, pp. 147–160

[15] Sarrouy, O., Totel, E., Jouga, B.: 'Building an application data behavior model for intrusion detection'. IFIP Annual Conf. on Data and Applications Security and Privacy, Montreal, QC, Canada, 2009, pp. 299–306

[16] Hofmeyr, S.A., Forrest, S., Somayaji, A.: 'Intrusion detection using sequences of system calls', *J. Comput. Secur.*, 1998, **6**, (3), pp. 151–180

[17] Creech, G., Hu, J.: 'A semantic approach to host-based intrusion detection systems using contiguous and discontinuous system call patterns', *IEEE Trans. Comput.*, 2014, **63**, (4), pp. 807–819

[18] Zimmer, C., Bhat, B., Mueller, F., *et al.*: 'Time-based intrusion detection in cyber-physical systems'. Proc. of the 1st ACM/IEEE Int. Conf. on Cyber-Physical Systems, Stockholm, Sweden, 2010, pp. 109–118

[19] Garfinkel, T., Rosenblum, M.: 'A virtual machine introspection based architecture for intrusion detection'. Network and Distributed System Security Symp. (NDSS), 2003, vol. 3, pp. 191–206

[20] Pfoh, J., Schneider, C., Eckert, C.: 'Nitro: hardware-based system call tracing for virtual machines'. Int. Workshop on Security, Tokyo, Japan, 2011, pp. 96–112

[21] Blunden, B.: '*The rootkit arsenal: escape and evasion in the dark corners of the system*' (Jones & Bartlett Publishers, USA, 2012)

[22] Davis, M., Bodmer, S., LeMasters, A.: '*Hacking exposed malware and rootkits*' (McGraw-Hill Inc., New York, NY, USA, 2010, 1st edn.)

[23] 'A hypervisor for rootkits', 2016. Available at: http://phrack.org/issues/69/15.html

[24] Perez-Botero, D., Szefer, J., Lee, R.B.: 'Characterizing hypervisor vulnerabilities in cloud computing servers'. Proc. of the 2013 Int. Workshop on Security in Cloud Computing, Hangzhou, China, 2013, pp. 3–10

[25] Tamrakar, S., Liu, J., Paverd, A., *et al.*: 'The circle game: scalable private membership test using trusted hardware', 2016, arXiv preprint arXiv: 1606.01655

[26] Kang, D.-K., Fuller, D., Honavar, V.: 'Learning classifiers for misuse and anomaly detection using a bag of system calls representation'. Proc. from the Sixth Annual IEEE SMC, Information Assurance Workshop (IAW'05 2005), West Point, NY, USA, USA, 2005, pp. 118–125

[27] Alarifi, S.S., Wolthusen, S.D.: 'Detecting anomalies in IaaS environments through virtual machine host system call analysis'. 2012 IEEE Int. Conf. for Internet Technology and Secured Transactions, London, United Kingdom, 2012, pp. 211–218

[28] Sultana, A., Hamou-Lhadj, A., Couture, M.: 'An improved hidden Markov model for anomaly detection using frequent common patterns'. 2012 IEEE Int. Conf. on Communications (ICC), Ottawa,canada, 2012, pp. 1113–1117

[29] Apap, F., Honig, A., Shlomo, H., *et al.*: 'System and methods for detecting intrusions in a computer system by monitoring operating system registry accesses', 2008. Available at: http://www.google.com/patents/US7448084

[30] Shabtai, A., Tenenboim-Chekina, L., Mimran, D., *et al.*: 'Mobile malware detection through analysis of deviations in application network behavior', *Comput. Secur.*, 2014, **43**, pp. 1–18

[31] Cavallaro, L., Sekar, R.: 'Taint-enhanced anomaly detection'. Int. Conf. on Information Systems Security, Guwahati, India, 2011, pp. 160–174

[32] Litty, L.: 'Architectural introspection and applications'. PhD dissertation, University of Toronto, 2010

[33] Borisaniya, B., Patel, D.: 'Evasion resistant intrusion detection framework at hypervisor layer in cloud'. Int. Conf. on Advances in Communication, Network, and Computing, Chennai, India, 2014, pp. 748–756
[34] 'Intel® 64 and ia-32 architectures software developer's manual'. Available at: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf
[35] Zhang, F., Leach, K., Sun, K., *et al.*: 'Spectre: a dependable introspection framework via system management mode'. 2013 43rd Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN), Washington, DC, USA, 2013, pp. 1–12
[36] Zhang, F., Leach, K., Stavrou, A., *et al.*: 'Using hardware features for increased debugging transparency'. 2015 IEEE Symp. on Security and Privacy (SP), San Jose, California, 2015, pp. 55–69
[37] Coreboot: Available at: https://www.coreboot.org/
[38] Zhang, F., Wang, H., Leach, K., *et al.*: 'A framework to secure peripherals at runtime'. European Symp. on Research in Computer Security (ESORICS 2014), Wroclaw - Poland, 2014, pp. 219–238
[39] Sielken, R.: '*Application intrusion detection: [technical report cs-99-17]*' (Department of Computer Science, University of Virginia, 1999)
[40] Razzaq, A., Latif, K., Ahmad, H.F., *et al.*: 'Semantic security against web application attacks', *Inf. Sci.*, 2014, **254**, pp. 19–38
[41] Miller, D.: 'Application intrusion detection'. Available at: https://www.blackhat.com/presentations/bh-federal-03/bh-fed-03-miller.pdf
[42] Zhu, Z.J., Zulkernine, M.: 'A model-based aspect-oriented framework for building intrusion-aware software systems', *Inf. Softw. Technol.*, 2009, **51**, (5), pp. 865–875
[43] Sallam, A.: 'New era of mega trends: hardware rooted security'. ARM TechCon Conf., Santa Clara, California, 2014. Available at: https://www.linkedin.com/pulse/20141015064547-2394400-the-new-era-of-mega-trends-hardware-rooted-security
[44] Sabt, M., Achemlal, M., Bouabdallah, A.: 'Trusted execution environment: what it is, and what it is not'. IEEE Trustcom BigDataSE ISPA, 2015, Helsinki, Finland, 2015, vol. 1, pp. 57–64
[45] Liu, Y., Xia, Y., Guan, H., *et al.*: 'Concurrent and consistent virtual machine introspection with hardware transactional memory'. 2014 IEEE 20th Int. Symp. on High Performance Computer Architecture (HPCA), Orlando, Florida, USA, 2014, pp. 416–427
[46] Pappas, V., Polychronakis, M., Keromytis, A.D.: 'Transparent rop exploit mitigation using indirect branch tracing'. USENIX Security Symp., Washington, USA, 2013, pp. 447–462
[47] Uhsadel, L., Georges, A., Verbauwhede, I.: 'Exploiting hardware performance counters'. IEEE 5th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC08 2008), Washington, DC, USA, 2008, pp. 59–67
[48] Maurice, C., Le Scouarnec, N., Neumann, C., *et al.*: 'Reverse engineering intel last-level cache complex addressing using performance counters', in '*Research in attacks, intrusions, and defenses*' (Springer, Cham, 2015), pp. 48–65
[49] Vogl, S., Eckert, C.: 'Using hardware performance events for instruction-level monitoring on the x86 architecture', 2012
[50] Demme, J., Maycock, M., Schmitz, J., *et al.*: 'On the feasibility of online malware detection with performance counters', *ACM SIGARCH Comput. Archit. News*, 2013, **41**, (3), pp. 559–570
[51] Bahador, M.B., Abadi, M., Tajoddin, A.: 'Hpcmalhunter: behavioral malware detection using hardware performance counters and singular value decomposition'. 2014 IEEE 4th Int. eConf. on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 2014, pp. 703–708
[52] Malone, C., Zahran, M., Karri, R.: 'Are hardware performance counters a cost effective way for integrity checking of programs'. Proc. of the Sixth ACM Workshop on Scalable Trusted Computing, Chicago, IL, USA, 2011, pp. 71–76
[53] Tang, A., Sethumadhavan, S., Stolfo, S.J.: 'Unsupervised anomaly-based malware detection using hardware features'. Research in Attacks, Intrusions and Defenses, Gothenburg, Sweden, 2014, pp. 109–129
[54] Wang, X., Karri, R.: 'Numchecker: detecting kernel control-flow modifying rootkits by using hardware performance counters'. 2013 IEEE 50th ACM/EDAC/IEEE Design Automation Conf. (DAC), San Francisco, CA, USA, 2013, pp. 1–7
[55] Jain, B., Baig, M.B., Zhang, D., *et al.*: 'Introspections on the semantic gap', *IEEE Secur. Priv.*, 2015, **13**, (2), pp. 48–55
[56] 'Intel® in target probe (ITP) ITP-XDP 3BR kit'. Available at: https://designintools.intel.com
[57] Ruan, X.: '*Platform embedded security technology revealed: safeguarding the future of computing with intel embedded security and management engine*' (Apress, Berkely, CA, USA, 2014)
[58] 'Vx heaven virus collection 2010-05-18'. Available at: http://academictorrents.com/details/34ebe49a48aa532deb9c0dd08a08a017aa04d810
[59] Serebrin, B., Hecht, D.: 'Virtualizing performance counters'. Euro-Par 2011: Parallel Processing Workshops, Bordeaux, France, 2011, pp. 223–233
[60] 'Hpcgnature'. Available at: https://github.com/amusavi/hpcgnature
[61] 'Perf: Linux profiling with performance counters', Available at: https://perf.wiki.kernel.org/index.php/Main_Page
[62] Gemplus, P.G.: 'Tool for injecting a shared object into a linux process', 2015. Available at: https://github.com/gaffe23/linux-inject
[63] 'Virusshare.com - because sharing is caring'. Available at: https://virusshare.com
[64] Ross, G.J.: 'Parametric and nonparametric sequential change detection in r: the cpm package', *J. Stat. Softw.*, 2013, **78**
[65] Bulej, L., Kalibera, T., Tuma, P.: 'Repeated results analysis for middleware regression benchmarking', *Perform. Eval.*, 2005, **60**, (1), pp. 345–358
[66] Rohr, M.: '*Workload-sensitive timing behavior analysis for fault localization in software systems*' (BoD–Books on Demand, 2015)
[67] Zaparanuks, D., Jovic, M., Hauswirth, M.: 'Accuracy of performance counter measurements'. 2009 IEEE Int. Symp. on Performance Analysis of Systems and Software (ISPASS 2009), Boston, MA, USA, 2009, pp. 23–32