MEDA: A Machine Emulation Detection Algorithm

Valerio Selis and Alan Marshall
Department of Electrical Engineering and Electronics, University of Liverpool, Brownlow Hill, L69 3GJ, Liverpool, U.K.

Keywords: IoT, M2M, Trust, Embedded Systems, Virtual Machines.

Abstract: Security in the Internet of Things (IoT) is now considered a priority, and trust in machine-to-machine (M2M) communications is expected to play a key role. This paper presents a mechanism to detect an emerging threat in M2M systems whereby an attacker may create multiple fake embedded machines using virtualized or emulated systems, in order to compromise either a targeted IoT device, or the M2M network. A new trust method is presented that is based on a characterisation of the behaviours of real embedded machines, and operates independently of their architectures and operating systems, in order to detect virtual and emulated systems. A range of tests designed to characterise embedded and virtual devices are presented, and the results underline the efficiency of the proposed solution for detecting these systems easily and quickly.

1 INTRODUCTION

The Internet of Things is a concept in which very large numbers of physical objects can be connected to the Internet. The IoT infrastructure requires an integration of several technologies and the objects (things) will be mostly connected wirelessly to the main infrastructure. In particular, the objects will be mostly based on embedded devices, such as sensors, smart-phones, etc. (Atzori et al., 2010). An essential role will be given to Machine-to-Machine (M2M) networks, this refers to communications among objects. In the future, M2M communications will mostly operate without human intervention.

When available, the main characteristics of these objects may be subdivided as follows:

- *MCU/CPU*: from embedded to high performance processor, such as ARM, MIPS, PowerPC, AVR, x86, x86-64, etc.: from few MHz to GHz, single core to multi core;
- *OS*: open source and proprietary, such as Linux based, Windows based, iOS based, Symbian, etc.;
- *Memory - data storage*: from few KB to TB;
- *Memory - RAM*: from few KB to GB;
- *Network interfaces*: from one to multiple interfaces at the same time, such as Wired, RFID, ZigBee, Bluetooth, Wi-Fi, Cellular, GPS, etc.;
- *Power*: from very low or zero power to high power consumption; battery and/or wired;
- *Type*: mobile or static.

It is possible to envision that in the future, networks of smart objects or "things" will manage and control parts of our lives in an autonomous way. They will operate in various application areas, such as healthcare, smart robots, cyber-transportation systems, manufacturing systems, smart home technologies, smart grids and building security (Chen et al., 2012; Lee et al., 2013). For this reason, securing the IoT must be a priority, before continuing to deploy it in the real world and on a larger scale.

An important factor in securing the IoT and M2M communications is trust. Nowadays there are various papers in the literature addressing how to manage trust among objects in IoT (Bao and Chen, 2012; Saied et al., 2013; Nitti et al., 2014). However, these do not consider how to trust the systems inside the objects. As shown in Figure 1, an attacker may create multiple fake embedded machines using virtualized or emulated systems in order to compromise the network or part of it. In this case, a real embedded device (A) will assume that it is communicating with other real embedded devices (B, C, D and E) and will trust them accordingly. This will lead to a security issue in the network and in particular for the data transmitted by "A" throughout the attacker(s).

The aim of this paper is to show the first stage of a possible detection mechanism to help trust agents in embedded devices to trust other systems and recognize a virtualized or emulated environment. The rest of this paper is organised as follows: In section 2 the

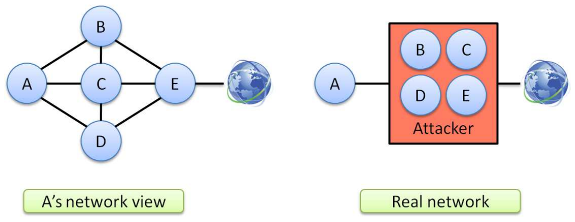

Figure 1: Representation of multiple fake embedded machines attack.

related work on detection of virtual and emulated systems is presented. In section 3 we describe the algorithm used to collect the behaviour information from real machines. The detection methods adopted and the results obtained from the characterisation tests are discussed in section 4. Finally in section 5, we present our conclusions and directions for future research on detecting virtual and emulated systems.

2 VIRTUALIZATION AND EMULATION DETECTION

Virtualized and emulated detection mechanisms have been mostly studied for x86/x64 architectures. The few works done so far are focused on the detection of Android-based emulated environments (Vidas and Christin, 2014; Jing et al., 2014). These detection mechanisms can be categorized as follows:

- CPU and memory tests: memory states and CPU registers and instructions are used to detect if a virtual machine (VM) is running
- Remote tests: a remote machine is used to collect information from network packets in order to understand the capabilities of another machine
- Timing tests: consist of using particular CPU instructions to perform a time analysis detection
- Fingerprinting tests: information about the system is collected, such as driver names, CPU ID, system registers, etc.

Rutkowska (Rutkowska, 2004) introduced a simple mechanism, called “The Red Pill”, to detect if a system was running on a virtual machine, simply by using the SIDT (Store Interrupt Descriptor Table) instruction to access to the IDT register. The author noted that the value of this register was different in real machine and virtual machine, because in an x86 CPU there is only one IDT register for every OS running in the system. However, this approach does not work in newer machines as they use multi-core processors and there is one IDT register for each of them.

In (Martignoni et al., 2009), the authors created an automated method to generate random red-pills. They used a CPU guided system to detect emulators such as QEMU and BOCHS for x86 architectures. This method consists of checking that the same instruction (red-pill) with the same CPU state will return the same state in the memory. If the result of the execution of a red-pill in the emulated CPU is different from the result in the real CPU, it will lead in a detection status. In (Shi et al., 2014) an enhanced red-pill method called cardinal pill is proposed. It consists of using all definitions present in the IA-32 manual to create pills and then check discrepancies between real CPU and emulated CPU. However, these mechanisms can fail to detect hypervisors that use the real CPU to execute the instructions, as, in these cases, the results would be the same. The authors in (Chen et al., 2008) stated that it is possible to use the method created by (Kohno et al., 2005) to detect virtualized hosts. The method proposed was able to detect the clock rate of a remote machine using the TCP timestamp option defined in RFC 1323 (Jacobson et al., 1992). However, this method cannot be used nowadays, as (Polcák et al., 2014) demonstrated, in Linux-like OS the timestamp present in the TCP packets is influenced by the application NTP, which provides an up to date timestamp to the system using time servers. Moreover, the method proposed in (Kohno et al., 2005) is not valid for remote machines that use a Windows based OS, because it does not use the TCP timestamp option by default, and the clock skew can be faked by the remote host observed (Polcák and Franková, 2014).

Another method used to detect a virtualized or emulated environment is time analysis. (Raffetseder et al., 2007) and (Jia-Bin et al., 2012) showed that by timing the access to control registers, such as CR0, CR2 and CR3, and the execution of a NOP instruction, it is possible to detect if the machine is real or not.

A fingerprinting mechanism uses specific hardware and software values to understand if there is a virtualized or emulated environment. (Chen et al., 2008; Raffetseder et al., 2007) suggested using the MAC address of a machine to obtain the vendor name. However, this can be faked very easily, for example by using an application like MAC-Changer (Ortega, 2013). More information can be obtained from drivers used for specific hardware devices, such as video card, network card, etc. Furthermore, registry keys or running applications give the opportunity to find a virtualized or emulated environment as underlined by (Chen et al., 2008) and (Jia-Bin et al., 2012). In some cases it is possible to check if the application is running in a virtual environment by accessing the virtual machine API as shown by (Quist and Smith, 2006).

There have only been a few studies related to the detection of embedded emulated environments, and these are focused at Android based devices. (Jing et al., 2014) proposed a heuristic detection mechanism by combining the Android API, Android system properties and the system hardware information with a detection time of 20 minutes. Moreover, (Vidas and Christin, 2014) used the same information collected in (Jing et al., 2014) with new information about the CPU and graphical performances to detect the Android's Dalvik VM.

As far as we are aware, there have been no general studies on the detection of virtual or emulated environments for embedded systems. In this work we propose a novel detection method that consists of checking the behaviours of embedded devices, virtual and emulated embedded systems. In this work we consider embedded CPU and MCU such as x86, MIPS and ARM, without considering specific information from the OS. CPU/MCU and memory tests, and timing tests are not used in this work because different embedded architectures use different registers and instruction sets. Moreover, remote tests, as previously described, are not applicable, considering that embedded systems based on Linux can use NTP to update their date and time. Additionally, we postulate that the fingerprinting detection method can be easily faked, in particular in fully emulated environments; nevertheless, as we describe, during some tests this may be a valid detection method. Finally, we believe that adopting our method it will be difficult for a virtual or emulated system to fake the behaviour results, because this uses the network stack in order to perform the characterization. In this case, the virtual or emulated system needs to check when a socket is created and handle it every time, because it can't know a priori that it was called by our method. This would lead to an increasing in the overall delay time and decrease in system performance. Additionally, there is a necessity to modify part of the kernel, which is not always possible.

3 CHARACTERISATION ALGORITHM

In order to perform the characterisation an algorithm was developed to collect information from real, virtualised and emulated environments. This consists pinging the localhost (127.0.0.1) in the system under consideration 1000 times, and for every ping collecting the ping response time (ms), the system timestamp (s) and the CPU/MCU usage (%), as shown in Figure 2.

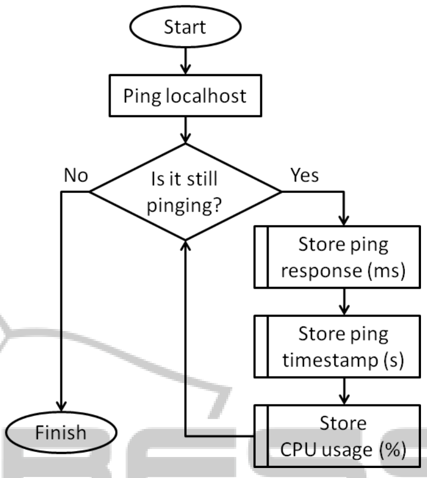

Figure 2: Characterisation algorithm flowchart.

The timestamp information was collected using the "date". The CPU/MCU usage was collected from "/proc/stat" or "iostat", depending on the OS used. We tested virtualized and emulated systems including: Android Emulator (Android Developers, 2014), Genymotion (Genymobile, 2014), GXemul (Gavare, 2014), OVPsim (Open Virtual Platform, 2014), QEMU (Bellard, 2005), VirtualBox (Oracle Corporation, 2014) and VMware Player (VMware Inc, 2015). All these were used with default configurations, along with a real machine used as reference (termed RM), which had the following characteristics:

- *OS*: Linux Mint 17 (qiana) with kernel 3.13.0-24-generic
- *CPU*: Intel(R) Core(TM) i3-4130 CPU @ 3.40GHz (4 cores)
- *RAM*: 7897 MiB

Next, we used as comparison, real embedded devices including: ALIX 6F2 (PC Engines GmbH, 2007), Google Nexus 5 and 7 (Google and LG Electronics, 2013; Google and Asus, 2012), Carambola (8devices, 2012), Arduino Yún (Arduino, 2013) and Raspberry Pi (Raspberry Pi Foundation, 2012). For each device, eight tests comprising the characterisation algorithm were performed by tuning the ping command with different options and stressing the CPU/MCU (whereby the CPU usage levels is maintained at 100%) as shown in Table 1.

In order to stress the CPU/MCU the "dd" command was used, with input data from urandom, if=/dev/urandom, and writing this data to the null de-

Table 1: List of characterisation tests performed.

Test#	Ping option (ping)	CPU/MCU stress
1	-c 1000	No
2	-c 1000	Yes
3	-c 1000 -i 0.2	No
4	-c 1000 -i 0.2	Yes
5	-c 1000 -s 20000	No
6	-c 1000 -s 20000	Yes
7	-c 1000 -s 20000 -i 0.2	No
8	-c 1000 -s 20000 -i 0.2	Yes

-c: stop after sending *n* ping packets
-i: wait *n* seconds between sending each packet
-s: specifies the number of data bytes to be sent

vice, of=/dev/null. In some characterisations, multiple instances of this program were executed to overload the CPU/MCU in multi-core devices. In these cases the information collected was analysed using different characterisation metrics for ping response times, timestamps and CPU/MCU usage levels as shown in Tables 2 and 3.

4 RESULTS OF CHARACTERISATION TESTS

The results obtained show the same behaviours for the real embedded devices in tests 1 to 4, however it was decided to exclude tests 5 to 8 as they do not give reliable results to identify virtual or embedded systems. The issues in identifying virtual or embedded systems in tests 5 to 8 were related to the ping packet size, whereby large-sized ping packets consume high computational resources which can cause problems in embedded devices with low processing power. Table 2 shows the range of behaviours concerning ping response times and timestamps for RM used during the tests. Information about the CPU/MCU levels are not shown for tests 2 and 4, because the CPU/MCU was under stress and its usage levels were always at 100%. Table 3 shows characterisation results for all real embedded devices (termed EM). In the rest of this paper we use the notations listed in Table 4 for the tests as well as the following notations: ping response time "P."; timestamp "T."; CPU/MCU level "C."; standard deviation "SD"; simulation "Sim.".

The detection method is based on the behaviours of RM and embedded machines as characterised in Tables 2 and 3. These ranges were used as threshold values to detect virtual or emulated systems. Let $\text{TMin}_X(\text{CM}_i)$ and $\text{TMax}_X(\text{CM}_i)$ be the minimum and maximum value in the range for the characterisation metric CM_i of X (RM or EM). Let $\text{T}(\text{CM}_i)$ be the CM_i value obtained from the target system. By considering RM, the virtual or embedded system is considered detected if $\text{T}(\text{CM}_i) < \text{TMin}_{RM}(\text{CM}_i)$. Moreover, it is considered better than RM if $\text{T}(\text{CM}_i) \ll \text{TMin}_{RM}(\text{CM}_i)$, this means that in some cases it is faster than the RM and/or the measurements obtained have a low error, i.e. P.SD close to 0, P.Total reduced by half or C.Mean less than 1% during tests 1 and 3. The machine emulation detection algorithm is based on the detection of illegitimate embedded devices. It uses the characterisation metrics based on EM and it is described in Figure 3. Considering EM, an illegitimate embedded device is considered detected if $\text{T}(\text{CM}_i) < \text{TMin}_{EM}(\text{CM}_i)$ or $\text{T}(\text{CM}_i) > \text{TMax}_{EM}(\text{CM}_i)$.

Input: set of characterisation metrics of the target system
Output: true or false

```
 for each $\text{CM}_i \in \text{CM}$:
   if(T($\text{CM}_i$) < $\text{TMin}_{EM}$($\text{CM}_i$) ||
        T($\text{CM}_i$) > $\text{TMax}_{EM}$($\text{CM}_i$) )
     return true        //illegitimate EM
 return false           //legitimate EM
```

Figure 3: Machine emulation detection algorithm.

Figures 4 and 5 show the results of the behaviour characterisations. These were obtained by combining the information gathered from the tests of the virtual and emulated systems using the RM as reference.

Figures 6 and 7 show the same results as Figures 4 and 5, but using the behaviours of embedded machines as reference. It can clearly be seen that when adopting the EM behaviours, the detection of virtual or emulated systems is significantly higher than when adopting the RM behaviours. It may also be observed that by considering all tests for the EM, the AE is more detectable than the GX2. From these results it is possible to observe that our solution detects every virtual and emulated system. Furthermore, it is possible to detect them by using only behaviours obtained from P.Total, P.Mean±SD and T.Total.

These results show that at least six virtual and emulated systems can be detected only considering RM behaviours and in particular by using P.Mean±SD. We observed that the Genymotion, OVPsim, VirtualBox and VMware behave close to, and in some cases better than RM. Moreover, Genymotion, VirtualBox and VMware are detectable using the fingerprinting test as shown in Table 5. This test was applied using detection values such as vbox, virtualbox, virtualized, oracle, innotek, intel, genuineintel, genymotion, vmware and their variants.

The final machine emulation detection algorithm is focused only on the detection of illegitimate em-

Table 2: Range of behaviours of the RM obtained from the characterisation tests.

Characterisation Metrics	Ping (ms) Tests 1 to 4	Timestamp (s) Tests 1 and 2	Timestamp (s) Tests 3 and 4	CPU/MCU usage (%) Tests 1 and 3
Min	0.011-0.021	0	0	1-6
Max	0.049-0.231	1	1	46-84
Max-Min	0.034-0.211	-	-	-
Total	34.347-64.067	199	999	-
Mean	0.034-0.064	0.199	0.999	13.627-14.306
Variance	0-0.001	0.159	0.001	40.070-64.292
Standard Deviation	0.004-0.033	0.399	0.032	6.330-8.018
Mean-Standard Deviation	0.026-0.033	-	-	-
Mean+Standard Deviation	0.038-0.097	-	-	-

Table 3: Range of behaviours of real embedded devices obtained from the characterisation tests.

Characterisation Metrics	Ping (ms) Tests 1 to 4	Timestamp (s) Tests 1 and 2	Timestamp (s) Tests 3 and 4	CPU/MCU usage (%) Tests 1 and 3
Min	0.067-0.193	0	0	0-75
Max	0.140-2.060	1-2	1-3	19-100
Max-Min	0.061-1.993	-	-	-
Total	99.064-288.117	199-201	999-1001	-
Mean	0.099-0.288	0.199-0.201	0.999-1.001	5.223-79.042
Variance	0-0.034	0.159-0.162	0.001-0.059	4.928-459.201
Standard Deviation	0.002-0.183	0.399-0.402	0.032-0.243	2.220-21.429
Mean-Standard Deviation	0.060-0.215	-	-	-
Mean+Standard Deviation	0.110-0.452	-	-	-

Table 4: List of notations of virtual and embedded systems for the tests.

System	Notation	Architecture	OS	System	Notation	Architecture	OS
Android Emulator	AE	ARMv7	Android 4.4.2	QEMU	Q1	MIPS	OpenWrt 12.09
Genymotion	GN	x86	Android 4.4.4	QEMU	Q2	MIPSel	OpenWrt 12.09
GXemul	GX1	MIPS	NetBSD 5.0.2	QEMU	Q3	ARMv6l	Raspberry Pi Debian
GXemul	GX2	MIPS	NetBSD 6.1.5	VirtualBox	VB	x86	OpenWrt 10.03
OVPsim	OVP	MIPS	Debian	VMware	VM	x86	OpenWrt 14.07

bedded devices as this is the aim of this work.

Comparing the information obtained from all the tests it is possible to observe that the results from tests 1 and 2 give a better detection than tests 3 and 4. In fact, by using the detection method developed for tests 1 and 2 it is possible to detect virtual and emulated systems in around 3 minutes. This is very important to minimise the power consumption in battery powered embedded devices and also be able to trust a system easily and quickly.

5 CONCLUSION

IoT is rapidly growing and M2M communications, provided by embedded devices, will have an important role in it. The primary goal must be to secure this communication by giving the ability to machines to trust each other. However, in order to trust each other these need also to communicate with trust agents with the capability to detect illegitimate embedded devices in the network. For this reason, we believe that the proposed algorithm is the first step to identify fake

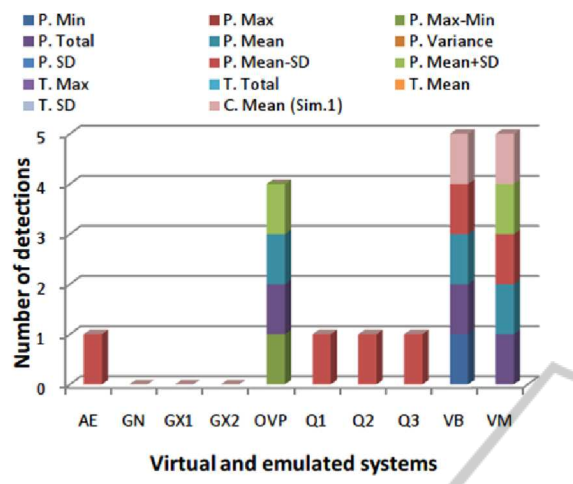

Figure 4: Behaviours detection using the reference machine for tests 1 and 2.

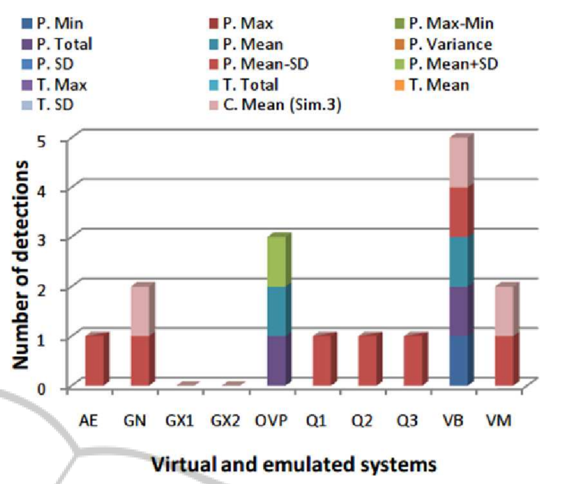

Figure 5: Behaviours detection using the reference machine for tests 3 and 4.

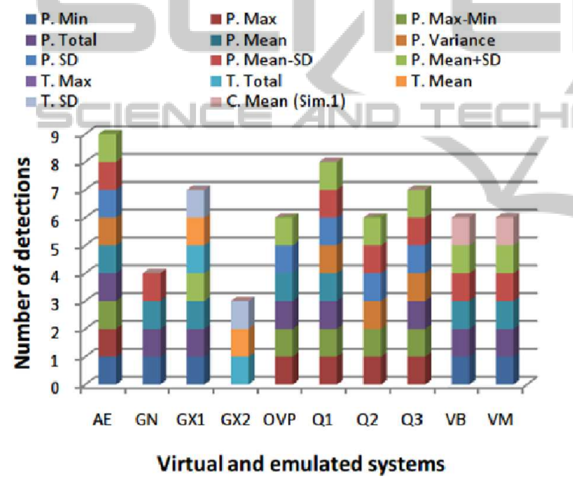

Figure 6: Behaviours detection using the real embedded machines for tests 1 and 2.

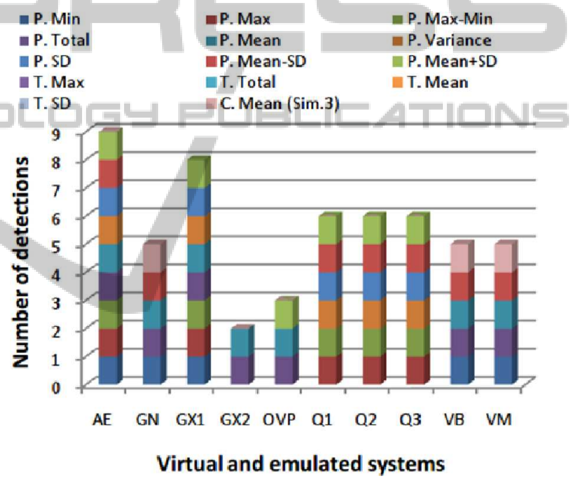

Figure 7: Behaviours detection using the real embedded machines for tests 3 and 4.

systems in order to create an effective trust among machines in IoT. A possible attacker would have to modify the TCP stack in the kernel in order to always return ping values in a specific range and with specific behaviours. Moreover, the attacker needs to know these values a priori, and a modification of the TCP stack may lead to other system issues. Also, any change in the kernel may lead to different timestamp values and this will be detected by the algorithm.

This work has demonstrated that the application of this method allows reliable detection of fake embedded machines that can compromise the network. This was done without using mechanisms applied to detect virtual and emulated systems in x86/x64 architectures, but by using a novel behaviour characterisation, which is independent of the architecture and the OS. In fact, other detection methods rely on the type of CPU architectures and/or the OS, thereby reducing their applicability in the IoT. The proposed method is not only effective, but it is also efficient and easily applicable to future IoT embedded devices.

Virtualized or emulated environments are normally used by anti-virus companies or researchers to study the behaviours or activity of malware in a system. The proposed method can be used by embedded malware, such as Chuck Norris botnet (Celeda et al., 2010) and Chameleon Wi-Fi virus (Milliken et al., 2013), in order to detect fake environments and in this case decide to change their behaviour. However, we think that detecting these fake systems will give great advantages for protecting the network.

Future research will involve improvements of this method and tests of a larger range of real embedded devices that are going to be part of IoT. Moreover, we

Table 5: Fingerprinting information about Genymotion, VirtualBox and VMware.

Artefacts from files or applications	Type of information	Genymotion	VirtualBox	VMware
/proc/cpuinfo	RM CPU characteristics	Yes	Yes	Yes
/proc/version	Linux version	Yes	No	No
/proc/misc	Virtual users	Yes	No	No
/proc/ioports	Virtual devices	Yes	No	No
/proc/kcore dmesg	Physical memory and system message	Yes	Yes	Yes
/sys/devices/pciXXXX:XX/XXXX:XX:XX.X/ XXXX:XX:XX.X/usb1/1-2/product /sys/devices/pciXXXX:XX/XXXX:XX:XX.X/ XXXX:XX:XX.X/usb1/1-2/configuration /sys/devices/pciXXXX:XX/XXXX:XX:XX.X/ XXXX:XX:XX.X/usb1/1-2/1-2:1.0/interface /proc/scsi/scsi	Hard disk, USB and CD-ROM devices	Yes	Yes	Yes
/sys/sys/devices/virtual/dmi/id/sys_vendor	System vendor	Yes	No	No
/sys/sys/devices/virtual/dmi/id/board_name	Board version	Yes	No	No
/sys/sys/devices/virtual/dmi/id/board_vendor	Board vendor	Yes	No	No
/sys/sys/devices/virtual/dmi/id/bios_vendor	BIOS vendor	Yes	No	No
/sys/firmware/acpi/tables/DSDT /sys/firmware/acpi/tables/FACP /sys/firmware/acpi/tables/SSDT	ACPI table information	Yes	Yes	Yes
/fstab.vbox86 /init.vbox86.rc	Virtual machine boot files	Yes	No	No
lsmod /system/lib/vboxsf.ko /system/lib/vboxguest.ko /system/lib/vboxvideo.ko	Virtual modules	Yes	No	No
/system/bin/androVM-prop /system/bin/androVM-vbox-sf /system/bin/androVM_setprop	Virtual machine software	Yes	No	No
ps /proc/XXX/mem	Virtual machine running applications	Yes	No	No
/system/build.prop	Android information	Yes	No	No
/system/etc/init.androVM.sh	Boot scripts	Yes	No	No

also plan to enhance the algorithms in order to more efficiently recognize virtual and emulated systems.

REFERENCES

8devices (2012). *Carambola*. [Online] Available from: http://www.8devices.com/carambola. [Accessed: 24 February 2015].

Android Developers (2014). *SDK Tools - Android Emulator*. [Online] Available from: http://developer.android.com/tools/help/emulator.html. [Accessed: 24 February 2015].

Arduino (2013). *Arduino Board Yún*. [Online] Available from: http://arduino.cc/en/Main/ArduinoBoardYun. [Accessed: 24 February 2015].

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805.

Bao, F. and Chen, I.-R. (2012). Dynamic trust management for Internet of Things applications. In *Proceedings of the 2012 international workshop on Self-aware internet of things*, pages 1–6. ACM.

Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46.

Celeda, P., Krejci, R., Vykopal, J., and Drasar, M. (2010). Embedded malware-an analysis of the Chuck Norris botnet. In *Computer Network Defense (EC2ND), 2010 European Conference on*, pages 3–10. IEEE.

Chen, M., Wan, J., and Li, F. (2012). Machine-to-machine communications. *KSII Transactions on Internet and Information Systems (TIIS)*, 6(2):480–497.

Chen, X., Andersen, J., Mao, Z. M., Bailey, M., and Nazario, J. (2008). Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 177–186. IEEE.

Gavare, A. (2014). *GXemul*. [Online] Available from: http://gxemul.sourceforge.net/gxemul-stable/doc/index.html. [Accessed: 24 February 2015].

Genymobile (2014). *Genymotion*. [Online] Available from: http://www.genymotion.com/. [Accessed: 24 February 2015].

Google and Asus (2012). *Nexus 7 (2012) Tech Specs (32GB + Mobile Data)*. [Online] Available from: https://support.google.com/nexus/answer/2841846?hl=en. [Accessed: 24 February 2015].

Google and LG Electronics (2013). *Nexus 5 Tech Specs*. [Online] Available from: https://support.google.com/nexus/answer/3467463?hl=en. [Accessed: 24 February 2015].

Jacobson, V., Braden, R., and Borman, D. (1992). TCP extensions for high performance. *RFC 1323*.

Jia-Bin, W., Yi-Feng, L., and Kai, C. (2012). Virtualization detection based on data fusion. In *Computer Science and Information Processing (CSIP), 2012 International Conference on*, pages 393–396. IEEE.

Jing, Y., Zhao, Z., Ahn, G.-J., and Hu, H. (2014). Morpheus: automatically generating heuristics to detect android emulators. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 216–225. ACM.

Kohno, T., Broido, A., and Claffy, K. C. (2005). Remote physical device fingerprinting. *Dependable and Secure Computing, IEEE Transactions on*, 2(2):93–108.

Lee, G. M., Crespi, N., Choi, J. K., and Boussard, M. (2013). Internet of Things. In *Evolution of Telecommunication Services*, pages 257–282. Springer.

Martignoni, L., Paleari, R., Roglia, G. F., and Bruschi, D. (2009). Testing CPU emulators. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, pages 261–272. ACM.

Milliken, J., Selis, V., and Marshall, A. (2013). Detection and analysis of the Chameleon WiFi access point virus. *EURASIP Journal on Information Security*, 2013(1):1–14.

Nitti, M., Girau, R., and Atzori, L. (2014). Trustworthiness management in the social Internet of Things. *Knowledge and Data Engineering, IEEE Transactions on*, 26(5):1253–1266.

Open Virtual Platform (2014). *OVPsim*. [Online] Available from: http://www.ovpworld.org/technology_ovpsim.php. [Accessed: 24 February 2015].

Oracle Corporation (2014). *VirtualBox*. [Online] Available from: https://www.virtualbox.org/. [Accessed: 24 February 2015].

Ortega, A. L. (2013). *MAC Changer*. [Online] Available from: http://www.gnu.org/software/macchanger. [Accessed: 24 February 2015].

PC Engines GmbH (2007). *ALIX 6F2 System Board*. [Online] Available from: http://www.pcengines.ch/alix6f2.htm. [Accessed: 24 February 2015].

Polcák, L. and Franková, B. (2014). On reliability of clockskew-based remote computer identification. In *International Conference on Security and Cryptography. SciTePress-Science and Technology Publications*.

Polcák, L., Jirásek, J., and Matousek, P. (2014). Comment on remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, (5):494–496.

Quist, D. and Smith, V. (2006). Further down the VM spiral-detection of full and partial emulation for IA-32 virtual machines. *Proceedings of the Defcon*, 14.

Raffetseder, T., Kruegel, C., and Kirda, E. (2007). Detecting system emulators. In *Information Security*, pages 1–18. Springer.

Raspberry Pi Foundation (2012). *Early versions of the Raspberry Pi Model B*. [Online] Available from: http://www.raspberrypi.org/documentation/hardware/raspberrypi/models/README.md#modelb. [Accessed: 24 February 2015].

Rutkowska, J. (2004). Red pill: Detect VMM using (almost) one CPU instruction. [Online] Available from: http://web.archive.org/web/20041130172213/http://invisiblethings.org/papers/redpill.html. [Accessed: 24 February 2015].

Saied, Y. B., Olivereau, A., Zeghlache, D., and Laurent, M. (2013). Trust management system design for the Internet of Things: A context-aware and multi-service approach. *Computers & Security*, 39:351–365.

Shi, H., Alwabel, A., and Mirkovic, J. (2014). Cardinal pill testing of system virtual machines. In *Proceedings of the 23rd USENIX conference on Security Symposium (SEC'14). USENIX Association, Berkeley, CA, USA*, pages 271–285.

Vidas, T. and Christin, N. (2014). Evading android runtime analysis via sandbox detection. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 447–458. ACM.

VMware Inc (2015). *VMware Player*. [Online] Available from: https://www.vmware.com/. [Accessed: 24 February 2015].