Magdeburger Journal zur Sicherheitsforschung

Gegründet 2011 | ISSN: 2192-4260
Herausgegeben von Stefan Schumacher
Erschienen im Magdeburger Institut für Sicherheitsforschung

This article appears in the special edition »In Depth Security – Proceedings of the DeepSec Conferences«.
Edited by Stefan Schumacher and René Pfeiffer

# Malicious Hypervisor Threat

## Phase Two: How to Catch the Hypervisor

**Mikhail Utin, PhD**

In this article we're addressing the matters discussed at DeepSec 2014 (Utin M. 2014) and 2016 (Utin M. 2016) including the current status of the Malicious Hypervisor (MH) project and the available information concerning it. The first part of our research - Phase 1 – was our analysis of a few publicly available documents concerning the MH threat, caused by the exploitation of virtualization and the out-of-band management vulnerabilities. The second part - Phase 2 – is about identifying Malicious Hypervisor activity, the discussion of discovery methods and, finally, the testing results of our HyperCatcher MH identification software. The matter of the MH threat is still evolving and we're planning on to address that in the future in Phase 3. Unfortunately, there is no end to the story of virtualization, vulnerabilities and threats. It has started by the implementation of mainframe OS virtualization in a PC environment. The technology was thus transferred from closed and secure mainframe architecture to an open and diverse Internet world without any thought of possible security implications.

Citation: Utin, M. (2017). Malicious Hypervisor Threat: Phase Two: How to Catch the Hypervisor. *Magdeburger Journal zur Sicherheitsforschung*, *13*, 754–771. Retrieved May 28, 2017, from http://www.sicherheitsforschung-magdeburg.de/uploads/journal/MJS_051_Utin_Hypervisor.pdf

# 1 The research history

The driving idea for our MH threats research has its roots in two really different sources - Stephen King's novel »Cell« (King 2005) and a Russian scientists blog post on a hackers' site describing an MH attack with the potential to completely destruct a computer system (Xakep.ru 2011). Besides all the futuristic components of Stephen King's novel, its idea of exploitation of mobile phone technology and networks rang the bell. In the novel the broadcast of a signal kills the human brain and creates millions of zombies… But who says that it is impossible to reach millions of computers at once and kill them, exploiting a security hole?

We completely depend on computing technology. Approximately 40 million servers worldwide do everything from growing vegetables to electronic banking. Everything is under computer control. Roughly one half of all servers belong to US organizations. How many of them malfunctioning would it take to disable first the US and then the world economy? Taking into account all the interdependencies of modern economy likely no more than 10%!

The fast development and production of information technology leaves security well behind, thus violating a fundamental principle – system development must start together with security development. A typical example is »cloud computing« (Utin, M. 2015). To plug a security hole in a widely used technology requires years of redevelopment and likely the search for new ideas. That complicates and over complicates computer systems without making them significantly secure. Multiplying functions, controls, codes, etc. increases the systems entropy raising the risk of various problems. Further below we will discuss how Intel IPMI's (Intelligent Platform Management Interface) (Scribed.com 2016) out-of-band management technology combined with virtualization (i.e. MH) may affect computers - thus turning the fiction of King's novel »Cell« into a grim reality.

Coincidentally, while reading »Cell«, we received a copy of a blog post of a Russian scientist– while working on his virtualization project , he found an embedded hypervisor in the Intel motherboards Baseboard Management Controller (BMC) system management software (Xakep.ru 2011). His post was a sort of outcry.

BMC is the implementation of Intel IPMI and exists in 99% of all servers. That exactly fitted the dooms day scenario of »Cell« – complete computer control up to its destruction plus possible unlimited distribution of malware via BMC BIOS embedded software to computers around the globe. The post was published on the blog at the end of 2011. It described events approximately between 2007 and 2010. The post is shocking, not only because it claims that for the first time an embedded hypervisor was found in the wild, but also that such software comes from the inside of motherboards of the main manufacturer of computer processors and chips – the Intel Corporation. The reason why this article was not known in the security research community was simply because the blog post was written in Russian. It was more than clear that such an embedded hypervisor cannot come out of nowhere, or could be the effort of a Black Hat team. Substantial efforts have been made, it is even possible that research on the matter has been published.

In his post the purpose of such an hypervisor or information about its payload – malicious or not - hasn't been taken into account. Anyway, the Russian scientist considered it a security threat. We, due to its silent existence, not disclosed by the company and the »parent« project (see below) - regard the hypervisor as a »malicious« component and will therefore refer to it as Malicious Hypervisor (MH).

We shortly found a publication of Michigan University (MU) (King, Samuel T. et al 2006) which describes the proof of concept of the development of malware, utilizing software virtualization. This malicious software modifies the boot record, reboots the system and modifies already installed OS (Windows or Linux) to run as a virtual guest OS. Finally, MH was developed to provide complete control over the guest OS, install additional virtual guests to run various exploitation tools and at the same time almost invisible from the above it now guest OS. Because of the time correlation and other details which we'll discuss below, we believe that the MH found in Intel motherboards has been initially developed in the MU lab.

At the same time, the idea of embedding an MH with whatever purpose in BMC BIOS seemed very risky and ineffective for its distribution. Thus we started to search for better »options of delivery«. We were not overly surprised when we found yet another MU research (Bonkoski, A. et al 2013) describing the vulnerability of out-of-band management software, which can be easily hacked over the management network interface. The article analyses various aspects of this problem, but to us the possibility of MH distribution to hundreds of thousands vulnerable computers over the Internet seems most important.

As the result of our 2013 – 2014 research our general understanding of the MH threat was as follows:

1. MH is the most advanced threat to high end computing systems utilizing virtualization.
2. There are no means reliably identifying the presence of MH.
3. MH instances have been found in the most secure place of high end computing systems – BMC BIOS.
4. MH instances can be easily distributed over Internet, by compromising vulnerable IPMI/BMC embedded software.

However, details of the research are important to understand the magnitude of the MH threat.

# 2 Details of MH Phase 1 research

Our research has been based on publicly available information. That includes three publications, which we briefly described above, and various related information of Internet research. We refer to these three publications as »Cases«:

Case #1 - Russian blog post (Xakep.ru 2011) »Chinese Add-ons: True Stories of virtualization, information security and computer spying« which we translated from Russian with the help of Google Translate and then edited manually. We tried to keep in line with the style of the unknown author, his technical considerations and emotions. All proves to us that the paper is genuine and not a crafted spammer's document to blame Intel Corporation in whatever is the name for such an activity of embedding and distribution. We will share our thoughts below. Events described in the post involved various people and organizations. Its URL has changed a couple of times since we verified the leaked text and the post may not be available anymore at the place where we found it at the time of this writing: https://xakep.ru/2011/12/26/58104/

Case #2 - US MU Virtual-Machine Based Rootkit (VMBR) research »SubVirt: Implementing malware with virtual machines" which was done approximately in 2005 to 2006 (Wang, Yi-Min et al 2006). The project was supported in part by grants of the National Science Foundation, CCR-0098229 and CCR-0219085, by ARDA grant NBCHC030104, by Intel and by Microsoft. The latter research team is listed as authors of the paper.

What is ARDA? It is now called DARPA (Defense Advanced Research Projects Agency). So, there are three VIP parties involved – the US government (defense agencies), IT software giant Microsoft and IT hardware giant Intel. It means the research was and is also »VIP«.

According to some Internet bloggers, the project information was initially available on the Microsoft site but has been removed later. We think it is now confidential or even classified.

Case #3 - US MU IPMI/BMC vulnerability research (Bonkoski A. et al 2013) »Illuminating the Security Issues Surrounding Lights-Out Server Management« conducted approximately in 2012 and 2013 (published 2013).

Interestingly, and that reflects the understanding of IT vendors and customers of the importance of the »system management network interface«, nobody gave a thought to its security until 2013. Well, let's say a vast majority of both vendors and customers didn't. How often vendors were issuing security updates for the system management BIOS? For old motherboards once in a while, say twice a year. It's a different matter now, when it comes to new systems – sometimes, they are updated each month. What has changed? This research has become well known and US CERT issued an IPMI related alert (US CERT 2013). However, there is another part in this security equation – vendors' personnel. The most of them are unaware of the threat and alert and do not perform system management software updates. Partially because physical server running a bunch of virtual systems should be offloaded of virtual guests, updated and then rebooted. This must be done for each physical server and causes system administrators a lot of headache, because it's a manual process and cannot be automated as user OS update.

Therefore, we think that because of the »human factor« the situation concerning server management vulnerability did not change significantly.

Our research process is illustrated in Fig. 1. It shows how we moved on historically from one case to another. How important is Case #1? First of all, it confirms that MH is not a theory but reality. What if anybody questioned the reality of the Russian scientist's findings? If we had »No Case #1« situation, we'd still have the same MH/IPMI threat and risks – virtualization vulnerability multiplied by IPMI/BMC vulnerability. The Case #2 of VMBR is relatively known but never discussed as a significant threat, although the MH is above system BIOS level and below OS and thus can alter any part or piece of OS and applications. So far, we have not came across any discussion of this threatening MH/IPMI combination.

## 2.1 Case #1 – Russian blog post

The Russian scientist's post describes how he found MH in Intel motherboards' BMC flash memory containing system management software. He identified it while developing his own virtualization software (hypervisor) for high performance computer systems. Kraftway, the customer concerned, is one of Russia's biggest IT companies, working closely with the Russian government. It supplied Intel motherboards with hardware assisted virtualization for the project.

While the development/testing system worked, the production system failed. The system has been hanging on the boot because another virtualization software was interfering with its own. That was what we now call »nested hypervisor«. The investigation finally revealed that the problem was located in the production motherboards, labeled »Assembled China«. Testing system boards had »Assembled Canada« labels. The scientist updated BMC embedded software in »Assembled China« motherboards from the Intel download site, and the production system finally worked.

He also observed that the process of improving MH in »Assembled China« motherboards caused less and less problems until the problem was completely gone as it was no BMC embedded virtualization software anymore. However, he was sure that now invisible software is still running on the motherboards, but working just fine as »nested hypervisor«. There were a few experiments described in the post to measure the execution time of specific commands. From his point of view, they proved that an MH existed and ran

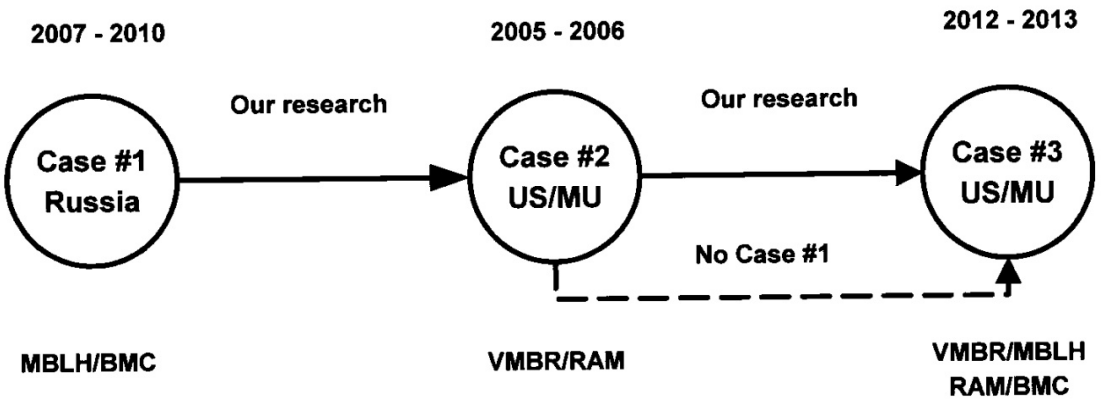


Fig 1. The process of Phase 1 MH research

his virtualization seamlessly. Unfortunately, there are no details provided in the blog post explaining the experiments and thus we cannot confirm provided results.

The scientist considered that the MH in motherboards represents a certain threat to the security of the Russian state. It also violates the law on encryption limitation because BMC embedded software is encrypted in the flash memory. Thus, he arranged two meetings with the Russian Federal Security Services (FSB) and gave a thorough presentation on MH technology and the threats and advantages of MH based exploits. He also gave a presentation for the information security team of GasProm (Russian natural gas production company) concerning the possibility of complete destruction of computer systems utilizing MH.

Unfortunately, his efforts to ring the MH threat alarm bell were in vain. He naively expected praises and openly expressed interest from FSB. But things don't work like that. The FSB definitely did not pass on the information and missed out on the opportunity to develop its own (or get an existing) MH solution to distribute silent control to points of interest.

So, basically, that was the end of the story. However, we would like to investigate some facts which appeared in the post but were not of interest to the author.

## 2.2 Case #1 – misleading labels

According to the post, the scientist worked with two shipments of motherboards – the one for testing, labeled as »Assembled Canada«, and the other one for production, labeled as »Assembled China«.

Our search through Intel Corporation's public information did not reveal any assembling factories in Canada and China. In fact, no such factories were in existence in 2007. Thus, both labels were fake. They have been placed on motherboards for a different purpose – possibly to distinguish whether embedded software was altered.

However, Intel Corporation does have a facility in Vancouver, Canada, which deals in flash memory and its embedded software.

Considering the known circumstantial evidence, we may think that both, the testing boards labeled »Assembled Canada« and the production boards labeled »Assembled China« came from Canada. Testing boards contained a pre-production version of BMC embedded software, so it's logical to assume them passing through the Vancouver facility. Thus, the »Canada« part of the label was correct. Concerning the production boards labeled »Assembled China« we consider two options:

1) the BMC software has been altered in the Vancouver facility or outside, at a »parallel« site, using the same kind of equipment, technology, development software, etc. So, we have either a company supported project of silently embedding a virtualization solution in its system management software (we consider such add-on as malicious though) or

2) a huge leak concerning confidential information of the company, including secret encryption keys, development software, management software code, equipment, etc. and the interception of the motherboards in question, altering the software and shipping it to the destination of interest.

Intel Corporation and other computer equipment manufacturers never disclosed their codes of system management software. It is considered strictly confidential. The software functionality is known in general, but, considering Case #1, it is not clear what else is embedded in a BMC/IPMI flash memory. We know that for the time being it was a hypervisor with unknown functionality. However, we do not know if this "BMC hypervisor« was a sort of pilot testing effort, if the hypervisor is still present in the BMC management software as a »feature«, and what its functions were or are.

## 2.3 Case #2 – MU »Virtual Machine (VM) Based Rootkit (VMBR)« research

The purpose of the project SubVirt was to create an ideal malware, and the MU team exploited virtualization technology for that purpose. Quote: »Our project, which is called SubVirt, shows how attackers can use virtual-machine technology to address the limitations of current malware and rootkits. We show how attackers can install a Virtual-Machine Monitor (VMM) underneath an existing operating system and use that VMM to host arbitrary malicious software. The resulting malware, which we call a Virtual Machine Based Rootkit (VMBR), exercises qualitatively more control than current malware, supports general purpose functionality, yet can completely hide all its state and activity from intrusion detection systems running in the target operating system and applications.«

The purpose of the research is clear, but there remains a question – how does the ideal malware fit in with the business of the project sponsors (NSF, Microsoft and Intel)? So far, we never heard of any of them being involved in any hacking activity. If that still holds true ARDA/DARPA would be the only interested party.

What have MU team done to attract defense agencies?

1. They used so called Virtual Machine Introspection (VMI) techniques enabling the VM service to understand and modify events within guest OS and its applications. That means complete control over the OS and its applications by malicious services within VMM or its malicious guests.
2. The Team designed and developed two proof-of-concept VMBRs based on VMware and Windows Virtual PC (currently obsolete). VMBR installs itself beneath the originally installed target OS and then runs the OS as guest. The installation requires the modification of the Master Boot Record to boot VMBR before the OS and then the reboot of the system. The VMBR is stored on free space on the system hard drive. All these operations require administrative privileges.
3. VMBR has malicious services as payload. However, to avoid detection by the target OS, VMBR installs attacking virtual host and use its OS to run malicious services. There are two classes of services. The first class does not communicate with the target OS. For instance, spam relays, DoS zombie agent, etc. The second class observes data or events in the target system and applications utilizing VMI. Examples are key logging, network packets capture, etc.
4. In general the research considers two ways of MH identification – from above VMBR and below. The research does not really consider a solution to identify VMBR from below, with the exception that yet another »Security Hypervisor (SH)« is installed before VMBR. However, such SH software should come from an even lower level, which is system management software in … BMC. So, we are getting exactly the same scenario that was later identified in the Russian case, in which the scientist was trying to install his hypervisor while it was already running MH from BMC.

   Finally, the SH concept has been used to implement a »Secure Boot« system.

   Does that mean that the purpose of the MH embedded in BMC in Case #1 is meant as protection from another MH installation? However, SH is not protected from the »rootkit« coming from the same level – via the out-of-band network management interface. That possibly makes the exploitation even easier – the hypervisor is legitimately installed and just needs to be hacked …

   Concerning the identification from the above, the research generally considers that MH consumes computer resources (CPU, memory, disks, I/O devices) thus MH activity can be identified. The problem is that MH can intercept identification activity and alter the system's status or hide its activity.
5. Future trends towards hardware assisted virtualization were analyzed. Execution of virtualization commands at CPU level will make it more difficult to identify MH activity. That also includes inserting VMBR (i.e. MH) between VMM and an already virtualized OS. This solution – VMM and several virtual OS/hosts - is currently dominating and possibly is used in 99% of IT systems. It is the authors opinion that VMM running as SH may help to improve the protection against VMBR, but that has yet to be proved.

## 2.4 Case #3 – Michigan University »Illuminating the Security Issues Surrounding Lights-Out Server Management« research

Here is the quote explaining the purpose of the research (Bonkoski, A. 2013): »This paper examines the security implications of the Intelligent Platform Management Interface (IPMI), which is implemented on server motherboards using an embedded Baseboard Management Controller (BMC).We consider the threats posed by an incorrectly implemented IPMI and present evidence that IPMI vulnerabilities may be widespread. We analyze a major OEM's IPMI implementation and discover that it is riddled with textbook vulnerabilities, some of which would allow a remote attacker to gain root access to the BMC and potentially take control of the host system. Using data from Internet-wide scans, we find that there are at least 100,000 IPMI-enabled servers (across three large vendors) running on publicly accessible IP addresses, contrary to recommended best practice.«

»…at least 100,000 IPMI-enabled servers…« What is the reason for such insecurity? In our opinion there

are two reasons.

Firstly, vendors traditionally considered the interface of the management network as the »backyard« of the system. All attention has been focused on securing the data interface and the frontend system. The backend was not really important ... While the frontend had an automated update system, the backend had not. When the frontend had an OS firewall, the backend had no such thing. The backend OS is Linux, and it's a »bare bone« system minimized to take as little as possible of BMC flash memory. While standard Linux systems run SELinux and an internal firewall, a system management OS does not. However, system management software includes various GUI applications accessible via HTTP/HTTPS. While Linux may be considered a relatively secure code, applications, and HTTP servers in particular, are traditionally full of bugs. In short, this is the first reason – vendors didn't pay appropriate attention to the backend system security.

The second reason is more fundamental. We cannot agree with the statement of Case #3 »... threats posed by an incorrectly implemented IPMI...«. It is not about implementation. IPMI is a standard, which should include, but has nothing related to security. It is an IT solution for system management, and from its draft until now DOES NOT INCLUDE SECURITY parts. In short, IPMI is the result of a very common IT approach - first comes system design and development and then, eventually, somebody may think about security. We have IPMI »technology« vulnerability when the IT technology solution lifecycle does not include security components. We will consider that below.

### 2.5 Cases Summary

By itself MH based on MU VMBR is very threatening, being capable to intrude a computer system, operate silently, while staying completely invisible for available anti-malware tools. It can be tasked to destroy the system as well. However, the penetration in working OS and the installation of malicious software requires administrator/root account level. Not a problem these days. Elevation of privileges (or privilege escalation) attacks have been around for a long time. To stay in the system long term MH needs to modify the boot process and save itself on the systems hard drive. Modern protection like »Secure Boot« may make that more difficult.

A more promising approach is the utilization of IPMI vulnerability by exploiting BMC and its memory for the MH penetration. Research (Bonkoski, A. 2013) proved that hundreds of thousand of systems of various vendors could be easy exploited and malicious software like MH could be installed in BMC RAM and then likely in flash memory. The vulnerability has an associated CERT alert TA13-207A (US CERT 2013), which describes the risk as follows (quote): »Attackers can use IPMI to essentially gain physical-level access to the server. An attacker can reboot the system, install a new operating system, or compromise data, bypassing any operating system controls.«

MU VMBR (Case #2) proved the concept of the most advanced malware. MU IPMI/BMC (Case #3) research explains how such malware can be distributed. Therefore, we have to conisder an extremely dangerous combination of two threats – MH and IPMI/BMC. The latter enables massive delivery and installation of extremely dangerous malicious software. Such software could be developed within one to two years by a qualified team of three or four people and distributed to millions of computers. Considering the fact of cyber terrorism, and taken into account that terrorist groups often show a very high level of qualification, it seems possible that an MH/IPMI cyber terrorism solution will be developed. This may bring us to the scenario of "Cell".

While IPMI/BMC software vulnerabilities may help to get the MH into a computer system, the management interface can be successfully used for the silent download of MH into BMC's flash memory, instead of having it preinstalled in motherboards as in the Russian Case #1. That could be done by a backdoor embedded in BMC, which silently downloads BMC BIOS with MH. The utilization of each method actually depends on the purpose of an MH installation. If the MH is used for »protection«, it may be embedded in each motherboard, and if for unidentified purposes, then it can be downloaded.

## 3 Correlation between Case #1 and Case #2 hypervisors

First of all, the MU VMBR research was and still is the most significant in the field of the MH, providing both concepts and practical examples of implementation. Its code may be definitely used in any future MH development.

And there is definitely a time correlation between the publishing of Michigan University's VMBR research (2006) and the Russian Case #1 of 2007 – 2008. One or two years should be sufficient to organize the process of embedding MH in motherboards and to develop a version working with hardware assisted virtualization.

One of VMBR project sponsors was Intel Corporation and thus had access to the entire research and software code. The latter, supposedly, was finally embedded in the BMC software.

The VMBR research considers only one out of all possible scenarios, for how to install malicious VMBR in an already virtualized customer environment having the customers VMM (hypervisor) and virtual guest OS. The VMBR is installed *between* VMM and guest OS instead of *below* VMM. That happened in Russian Case #1- the already running embedded hypervisor (in our terms - MH) installed a new customer hypervisor above itself.

The installation of a hypervisor above VMM may hap-

pen only if this VMM has been installed as a »security hypervisor« with system management privileges *from BMC software*.

VMBR in MU Case #2 was developed as a *software virtualization component* and thus as not being able to work with hardware assisted virtualization provided by new CPUs. Initially, the MH in Russian Case #1 was also not able to work with hardware assisted virtualization properly and to seamlessly support »nested hypervisors«. That was eventually fixed.

Considering such correlation, it is very likely that VMBR code has been used to develop the MH identified in the Russian Case #1.

# 4 How many MH instances do exist?

We assume that there are at least two MH instances – the original VMBR code instance working as a software virtualization component only, and a newer version working with hardware assisted virtualization, which has been identified in the Russian Case #1. Both can be used successfully. The first MH will work with computers not having virtualization support and the second with high end workstations and servers. Thus two instances should cover all computers in use which has been manufactured within the last ten years and the ones which will be manufactured within the next five years. Of course, changes in hardware assisted virtualization will have to be addressed, but such an upgrade is not a major issue.

There is also a high likelihood of one more MH instance, one »Made in Russia«. In his blog post the scientist described his meetings with the Russian government authorities from FSB (former KGB) and him informing them about the treats and advantages of this particular malware. He was disappointed that the secret service people expressed no interest and dismissed the threat. And then there were his two presentations at GasProm, which, although technically successful, seemed to have failed to raise attention as well. These experiences basically were his main reason to publish the case. However, the scientist was naive to expect the FSB or other security professionals to shake hands and promise to work with him on the MH case. He released all the information they needed and thus became irrelevant to the future plans of secret services.

It seems likely that a new instance of the MH was developed in Russia within a couple of years or less, and available around 2011 to 2012. But if that's the case, we should not expect anything like the MU VMBR research publication. There is this one rule of the Russian security services - classify all just in a case.

# 5 Technology vulnerability vs. software vulnerability

People who are involved in security research, respectively the search of software code vulnerabilities, know how difficult it is to discuss with software vendors code weaknesses, which are not pure *bugs* but often referred to by vendors as »*features*«. Such «*features*« affect the software security as well, but there is a strong resistance to admit this fact. The reason for such cover-ups is pretty obvious – vendors want to keep face. On the other hand, the reasons for having bugs and *features* in software code are complex:

1. Competition – success on the IT marketplace requires speeding-up the development and testing of code and that obviously generates more bugs and flaws in the design;
2. Complexity of IT software – complexity requires more time for design and development, thus affects competition (see above);
3. Globalization – to cut expenses and get ahead in competition, companies outsource design and development and export cheap labor from developing countries. That may cut expenses but at the same time may cause more bugs, simply because cheap labor often means less quality, caused by less experience and education. Thus, we are getting a lower quality software product with more *bugs* and *features.*
4. Lawlessness – there is no general information security law in the US, which would require a Security Development Lifecycle for all IT systems like the Certification and Accreditation (C&A) process for the US government information systems (which is often violated anyway). Many thanks to various IT lobbies. It is up to the software vendor whether to implement security measures and to which extent.

No one takes responsibility for bad code at all. Every software on the market states this in its legal disclaimer.

Below we are discussing *features* which cause significant security vulnerabilities in almost all server class computers.

As we discussed, the above threats employ two technologies – virtualization and out-of-band management. VMBR or any other MH can affect only computer systems running virtualization solutions. If there is no virtualization, you're not at risk. Quite the same can be said when it comes to out-of-band management – if there is no IPMI implementation there is no IPMI/BMC problem with its associated threats and risks. Below we'll prove that when we talk about such problems we actually talk about *the vulnerability of technology*. Because this point of view was objected by the Intel Corporation Production Security we need to prove our case.

Here are two definitions of *vulnerability* from Wikipe-

dia (Wikipedia 2016a):

1. General - »*Vulnerability* refers to the inability (of a system or a unit) to withstand the effects of a hostile environment.« To put it simply – vulnerability is the inability to resist a threat.
2. In computing - »In *computer security* vulnerability is a weakness which allows an attacker to reduce a system's information assurance. Vulnerability is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, and attacker capability to exploit the flaw.«

The latter definition does not mean *software bugs* and related vulnerabilities. Computer system vulnerability may be caused not by bugs but by system *features*, by incorrectly designed functionalities. In our MH case we do not talk about *software bugs* as well. MH is a *computer technology vulnerability*. And that's what's usually not discussed.

Here is our definition of *computer technology vulnerability: It is the inability of a technology implemented in a computer system to resist a threat when there is the technology flaw, attacker access to the flaw and attacker capability to exploit the flaw*.

We can formulate that MH exploits modern operating systems inability to resist involuntary virtualization after OS (or hypervisor) is installed and functioning. Operating systems do not have protecting mechanisms against malicious virtualization, neither does the computer system itself. Basically the flaw is the lack of any protection mechanisms designed to guard against malicious virtualization. Old mainframe systems had such protection – there was no access on the system level to install software from an outside source.

Similarly to *computer security* vulnerability, virtualization vulnerability is the intersection of three elements - there is modern OS susceptibility to MH inflicted virtualization, MH administrator level access to OS gained by various means, and then the MH's capability to use various exploitation tools.

The case of IPMI/BMC is also very similar – it is an IPMI technology vulnerability of seamless and unprotected system level access to computer resources, which is usually utilized by the computer management system. And then there's also the BMC technology vulnerability of unprotected implementation of system management software embedded in BMC like in the Russian Case #1. Here, again, we have both technologies susceptibility, attacker access to first management system and then to computer internals, and, finally a wide range of exploitation techniques and tools. CERT alert TA13-207A (US CERT 2013) describes exactly the same scenario.

The vulnerability of software code has to be addressed by the software vendor, if the vendor agrees that such a vulnerability exists. But what about *technology vulnerability*? In particular, when such technology has been used for years and is likely to be used in the near future.

We think that vendors should take care of such vulnerabilities, especially when their technology has an underlying standard like IPMI.

Microsoft was one of the sponsors of the MU VMBR research, its team is even listed as a research participant in the paper (King, Samuel T. et al 2006). That means that the discovered vulnerability of its Windows OS and underlying computing technology was known to Microsoft since 2006. Since then, Microsoft developed its own virtualization solution Hyper-V, which may be susceptible to an MH attack. However, we have never seen a Microsoft publication addressing virtualization vulnerability. It looks like all major IT vendors share the same opinion, expressed by Microsoft Hyper-V Program Manager Ben Armstrong (Mackie, Kurt 2013) that UEFI (Unified Extensible Firmware Interface) and »secure boot« will protect from rootkit (i.e. MH) attacks.

We do not share such optimism as expressed by IT vendors' management, considering the following:

1. A MH initiated from BMC can run at so called »ring -2« System Management Mode (SMM) privilege, which is the highest in the system (hardware virtualization has »ring -1« and kernel – »ring 0«). Therefore it can alter any information used for »secure boot«.
2. So far we have not seen any publications of real life UEFI and »secure boot« testing utilizing the MH instance for attacking. Such a publication would support the vendors' security claims. For instance, a hypervisor embedded in Intel motherboards could be used ...
3. There are two papers of Invisible Things Lab Company which explain attacking Intel TXT (Trusted Execution Technology) using two different methods – SMM flaw attack (Wojtczuk, Rafal 2009) and exploiting a bug in SINIT module (Wojtczuk, Rafal 2011).
4. We had a problem running backup software when we installed it in »secure boot«. It was not a software problem, but for reasons unknown one file signature was not recognized while signatures by the same vendor for other files were recognized. The vendor advised to disable »secure boot« while installing the software. Such experience does not encourage our trust in the boot solution. It may be buggy or has a »*feature*« - as we discussed above.

The second IT VIP sponsor - Intel Corporation - also never issued a report about working on fixing a vulnerability of its hardware assisted virtualization and IPMI/BMC vulnerability. Now, after the acquisition of McAfee, the company is involved in the information security business as well.

Intel Corporation is the major player on the computing technology market and provides the following products, which are related to our case:

- A line of CPUs (approximately 94% or CPU market)
- Chip sets for motherboards

- Motherboards
- CPU hardware assisted virtualization solution (Intel VT)
- Computer System Management Software (CSMS)
- OS to run CSMS (Linux)

Intel was the key player in the IPMI standard development. Its first version was published in September 1998 (Scribed.com. 2016). Intel also implemented its own version of server management software utilizing IPMI/BMC.

We considered that it is necessary for us to know about the corporations position concerning the *technology vulnerabilities* in question (virtualization and IPMI/BMC). We sent an official letter to Intel management, explaining our position on both vulnerabilities and our concerns over existing MH instances and the possibility of yet another development designed for cyber-terrorism purposes.

We did not have any illusions about if the IT giant will agree that vulnerabilities exist and that the cumulative effect of exploiting both vulnerabilities could be devastating. All we requested in our official letter was an explanation of the corporations opinion. Intel Production Security Team Lead (STL) responded and we had a few emails exchanged and one phone conversation. While we tried to encourage Intel to be involved in the research of possible solutions, our technical discussion with STL went to questionable technical statements like »We address BMC threats by disabling the management network interface *by default*« or » A future version of IPMI (Redfish) will have security«. We responded to the first statement that it undermines the purpose of such management which is *the remote control* of servers. We cannot imagine server administrators moving into a data center and managing servers one by one over the local serial line interface! Finally, we received an email basically dismissing the *technology vulnerabilities* case. The complete text is quoted below.

Quote:

> "Hi Mikhail,
>
> We'd like to thank you for raising your concerns to Intel regarding Intelligent Platform Management Interface (IPMI) and Virtualization Technology (VT). We take the security of our products and infrastructure seriously and work continuously on the security of both. We have carefully reviewed all the information you have provided as well as the resources you have directed us to.
>
> Intel published the first IPMI specification in 1998. Since that time it has worked with many other companies to extend the specification. As you know, security depends on how the specification is implemented and deployed by the system owner. As you've pointed out there are risks if vulnerabilities exist in the implementation or if the system is not deployed properly. At this point we have not received any new information from you that Intel implementations of IPMI have a vulnerability.
>
> Many parties in the industry, including Intel, provide detailed guidance regarding the proper use of this technology in order to help ensure systems follow good security practices. At this point we have not received any new information that VT has a vulnerability. If you are aware of one please do let us know. Additionally, if you'd like us to facilitate a discussion between you and a system provider for whom you've identified a vulnerability we'd be happy to make the connection.
>
> Regards,
>
> Intel PSIRT«

In short, Intel Production Security's opinion is:

1. (1) **There is no IPMI vulnerability. Problems are caused by incorrect implementation**.

   In fact, US CERT issued its *alert not about a bug but about IPMI/BMC* (US CERT 2013) »Alert TA13-207A. The Risks of Using the Intelligent Platform Management Interface (IPMI). US CERT, July 26, 2013«. And, as we wrote above, the IPMI standard, which explains implementation, does not include any consideration of security.
2. **There is no vulnerability in VT**. But the Russian Case is an example of such a vulnerability – The Russian scientist's post claims that his hypervisor has been installed above the hypervisor from the Intel motherboard, which acted as MH. This is a pure exploitation of a user virtualization solution which is based on Intel VT. During our email exchange we forwarded the post of the Russian scientist (Xakep.ru 2011) to the STL of Intel Production to make sure the company is aware of such public information in case it wants to object or discuss it.

But our opinion concerning *technology vulnerabilities* in Intel solutions and then products was dismissed.

# 6 Phase 2 MH research - detection

IT industry's various publications frequently mention the »rootkit« hypervisor (i.e. MH) as a threat. However, there is almost nothing to propose as a protective measure except »secure boot« and implementing its technologies like UEFI and TXT. Our doubts about this procedure have been expressed above. The most important doubt, so far, is the lack of testing of this sort of protection. It seems that for now we need to take care of security ourselves.

In short, that was our understanding after Phase 1 and its presentation (Utin, M. 2014). Our resources were limited and we cannot develop a "secure hypervisor« embedded in BMC BIOS. What we realistically were capable of was to start MH detection.

## 6.1 Detection methods

As we described above, MU VMBR research also includes general research on the identification methods. A realistic method is running MH identification software above the hypervisor, for instance within the installed operating system and catch hypervisor specific activity. One of the described ideas was the hypervisor detection by resources consumption (King, Samuel T. 2006). In the Russian Case #1 research the scientist used the same method and measured the increase of commands' execution time (Xakep.ru 2011).

There is thorough research on the matter of hypervisor identification including the classification and analysis (Korkin, Igor 2015) of various methods. The author considers Signature, Behavior, Trusted Hypervisor and, as above, Time based identification. Within the latter method he proposes »Detection by Unconditionally Interrupted Instructions« through the method of Instruction Execution Time. It utilizes the measurement of the execution of a number of CPUID instructions in Time Stamp hardware counter (TSC).

The main problem is that a MH, which completely controls virtualized OS via VMI, can identify such intelligence activity and either block or alter its results. This is our main concern with all methods listed above. The author then is proving by experiments and collected statistics that his proposed advanced method is resistant to the MHs attempts to alter the mean execution time.

Experiments were performed on six PCs having different CPU models and running three Windows OS (Windows 7, XP and Live CD XP). The team also used their own design of a hypervisor »with secure system monitoring functions« and the Acronis Disk Director hypervisor as MH.

The most important question, however, is the method's practicality. The paper describes thorough experiments collecting results over ten days to gain statistically stable results by running the identification software within the general purpose OS, which was Windows. We have seen similar results of increasing deviation in Linux OS as well. Utilization of computer resources is always a non-stationary random process. There are two different methods to deal with this: 1) by running the identification software for very long time to average results (as above) or 2) for a short time while processes causing non-stationary fluctuations are sleeping. It is usually possible to identify silent time for servers and then to run a detection software. To do that, the minimal time for statistically stable results should be known. For the experiments above, unfortunately, the minimal time has not been identified.

## 6.2 Our ideas and methods

From the very beginning of our research, we put aside what may be considered as the »mainstream efforts« of MH identification, which we described above. We finally managed to significantly change the time based method to make it simple and effective. Our methods, which we'll partially describe below, are US patent pending. The following, however, is not a complete description of what we have done and what exactly was implemented. We would like to challenge the security community to find different ways of implementing and innovating our approach – »no mainstream« solutions. Our main goal is to encourage the community to continue the research and possibly to find better methods following the requirement of *simplicity and practicality*.

From the very beginning we made *simplicity and practicality* the cornerstones of our identification method. Basic ideas were very simple:

1. If a hypervisor exists in the system, it should consume additional CPU resources; therefore, we can use a performance based identification method of Time Difference Identification (TDI). We expected such additional consumption (overhead) around 1% of what is needed to run an OS without an underlying hypervisor. This estimation is based on common sense. 10% would be absurd for a modern virtualization environment when one hypervisor usually runs around 10 virtualized OS. However an additional cosumption of 0.1% seems simply impossible, because a hypervisor is after all an OS and with all functionalities its resource consumption can't be that low. We would like to mention that, unfortunately, software vendors do not provide overhead values for a hypervisor's CPU.
2. As a software which emulates computer hardware, MH may react differently on resources requests (operations) coming from a user program compared to how the operating system would natively respond. As it has been identified, that causes random high level execution time deviation. We called this method »Deviation Difference Identification (DDI)«. Special operations will likely increase execution time as well and we can thus use the hybrid TDI+DDI method.
3. Our next step was to come up with an idea of how to identify a 1% difference in tests. We decided to use a 100% CPU utilization while running the identification software. In this case the hypervisor CPU overhead will create an increase in execution time. If our identification software execution time without hypervisor (clean system) is Tc, then with a hypervisor (Th) we will hopefully get $1.005Tc < Th < 1.01Tc$.
4. What to run as identification software? The main problem here is MH's possible capability to discover identification activity and thus alter identification results. The best way to avoid this is not to access any system points that the hypervisor utilizes in its activity. We opted for a application software for general purpose, performing various intensive calculations in computer memory, utilizing very basic CPU commands only, and not

special commands used in other methods including CPUID like in the research discussed above (Korkin, Igor 2015). The idea behind is to masquerade identification activity as well. Thus, the MH will be unable to recognize whether it is running a general purpose application software or one which does identification.

In short *we don't do identification by instructions execution time but rather have a program to execute, which does not directly cross the path of the MH.*

5. We will run our identification software multiple times to get more stable statistics. Thus the program should run long enough to execute a sufficiently large number of CPU commands but also short enough to permit its cycling and a reasonable final testing time.

## 6.3 Testing process

In the case of TDI we need to find a difference in execution time between the same system *with* and *without* a hypervisor. Thus, we need to have two test phases.

First phase - A clean system testing. It can be done either on the same model, when we're sure it's clean (for instance right after its production) or, if possible, by disabling virtualization via computer BIOS. Not all systems, however, support virtualization manual control via BIOS. Disabling virtualization support in BIOS, however, may be intercepted by the MH and considered as an identification attempt. The MH can then block changes but provide output as the virtualization was disabled.

Instead of a »clean« system we can use a statistical variant – considering that »clean« systems are more common and MH infected are unusual. We can then create a database of the testing of multiple systems' and compare results for the same model.

For TDI testing, we need to plan for a database supporting such testing and keeping results for all computer system models in use.

DDI testing, in general, also requires two phases if we are going to use the difference of execution time. However, we were able to discover a specific effect of virtualization which made it possible to test only once. This will be discussed below.

We decided to use a specially built Linux OS environment to run our identification software. The reason was to significantly decrease the deviation of testing results associated with general purpose OS running user applications in either Windows or Linux. Such testing requires the reboot of the currently running system and boot our software. We assume that correctly developed MH should sustain such a reboot and virtualize our booted OS and detection application. Or, this is also possible, to reboot after installing our software.

While general purpose OS (Linux, Windows, etc.) increases results deviation, such effect may be compensated by higher values of results in the methods of DDI or TDI+DDI.

Then we faced the question of which hypervisor we should use for the development testing. We understood that we can't test multiple hypervisor instances, simply because of our lack of time and resources. Thus we decided to use the most advanced and mature VMware hypervisor. Our results proved that the VMware code is very efficient – the time increase (hypervisor overhead) is only about 0.7%.

Here is our development testing environment:

- Two high end Lenovo notebook computers supporting Intel hardware virtualization
- VMware ESXi 5.5.0 as hypervisor software with VMKernel Build 2068190
- Desktop computer running VMware vSphere Client 5.5 to run our identification software
- Bootable Ubuntu Linux OS CD with HyperCatcher identification software.

We have done our first tests, which took significant time, utilizing general purpose Linux CentOS 6.x Live CD OS. However, the testing results after statistical filtration showed a high deviation and thus it was difficult to distinguish the time increase. Full featured OS runs a lot of processes including GUI, updates, security, etc. To decrease the deviation we switched to Linux CentOS 7.0 Minimal Installation without GUI and finally to a special build Ubuntu 14.04. The execution time deviation decreased approximately by the order of magnitude.

We finally developed a three steps testing procedure which we named »run«. Its first step is to execute basic identification software a few thousand times to decrease the deviation; this is a cycle. We then calculate the average value for all CPU cores - this is the result of the execution cycle. However, the deviation was still high and required the next step of statistical filtering. We did several cycles in a trial to find the average execution time value for this trial and the deviation. We then had to add yet another step in statistical filtration by executing several trials in one run. The average time value was used to calculate the time increase of TDI testing. The deviation proves that the execution time random value is inside three standard deviation intervals.

If the deviation is still high, it's possible to add a fourth step of statistical filtering by executing a few runs in this test. However, as of today we use the three steps filtration.

During the TDI development testing we identified that some operations significantly change results. Firstly,they increase the time difference. But the most significant change is in the deviation: it increases several times comparing to the deviation without the hypervisor in some trials but not always. Thus, our idea of forcing the hypervisor to change its behavior was correct. This testing is called, as we mentioned above, Deviation Difference Identification (DDI).

### 6.4 Testing results

Tables 1, 2 and 3 below represent such results. Each test used three steps filtration. Table 1 shows the result for execution time *without* hypervisor. We did only ten runs to accumulate statistics because the deviation is really low. Tables 2 and 3 represent the total of 40 runs for the testing *with a hypervisor*. We did 40 runs to show the behavior of time execution deviation. In Table 1 AvTc shows the average execution time and AvDc states the average standard deviation. In Table 2 and 3 AvTh shows the average execution time *with a hypervisor* and AvDh the average standard deviation.

#### 6.4.1 Hypervisor identification by time difference (TDI)

Therefore, in the test *without* hypervisor (Table 1) we have an average execution time of AvTc=5.5005 and an average deviation of AvDc=0.0008 (approximately 0.001)

The average execution time by four tests *with hypervisor* (Tables 2 and 3) is AvTh=5.5984

The time difference AvTh – AvTc=5.5984-5.5005=0.0979 or 1.8%.

Average deviation by four tests *with hypervisor:* AvDh=0.0202

The maximum sum of average deviations is AvDc + AvDh = 0.0212.

It is possible to identify the hypervisor by the TDI method because the time difference of 0.0979 is more than four times the maximum sum of average standard deviations: 0.0979 > 4x 0.0230.

#### 6.4.2 Hypervisor identification by deviation increase (DDI)

Each of the four tests show deviations which have a value of more than 0.01. That's more than ten times higher than the deviation *without* hypervisor AvDc=0.001:

- Test 1: there are 4 high deviation values (0.0312, 0.0190, 0.0337, 0.0423). The first high value is in the second run.
- Test 2: there are 7 high deviation values (0.0328, 0.0544, 0.0255. 0.0224, 0.0277, 0.0471, 0.0151). The first high value is in the first run.
- Test 3: there are 5 high deviation values (0.0283, 0.0649. 0.0189, 0.0153, 0.0399 ) and the first high value is in the first run
- Test 4: there are 5 high deviation values (0.0283, 0.0649, 0.0189, 0.0153, 0.0399) and the first high value is in the second run.

Therefore, statistically, based on the experiment above, we can say that an identification of high deviation value will always happen, highly likely within 1 – 5 runs. The probability that the first run will have a high value of deviation is 50%. The probability that the second run will have a first time high value is 75%, and the third run 87.5%. The probability that the MH will be found in at least one run in a 10-runs test is about 99.9%.

#### 6.4.3 Is there a hypervisor? The DDI testing report.

While we are not dismissing the value of TDI testing, we currently consider the DDI method as superior. Mostly because it does not require »clean« system results for comparison. Concerning the testing time, the test can be stopped when the first high time deviation is identified. You can not be sure when exactly, but it's likely this will occur within 4 – 5 runs.

Most of the values above are much higher than 0.01. The lowest value is 0.0151, that's 15 times more than the deviation without the hypervisor AvDc=0.001.

We can consider 0.01 as a boarder value. Any value higher means that a hypervisor is present. The following thresholds are suggested:

- Values lower than 0.002 should be considered as no hypervisor
- Between 0.002 and 0.005 - there is some likelihood that a hypervisor exists
- Between 0.005 than 0.01 a hypervisor is possible
- Above 0.01 – a hypervisor is identified

The drawing in Fig. 1 below shows a sample of deviation distribution in a DDI testing of 8 runs. The following screenshots Fig. 2 and Fig. 3 are results of two tests running HyperCatcher v.1.0. Each test had five runs. Yellow crosses show the runs' execution time deviation values and indicate if hypervisor was present. Blue crosses are trial deviations and for indication that the testing is in progress.

Screenshots were taken from VMware »mhhost« virtual machine console running MH identification software final production HyperCatcher version 1.0.

## 7 Conclusion for Phase 1 and Phase 2 MH research

1. The blog publication of the Russian Case #1 (Xakep.ru 2011) provided us with circumstantial evidence that a VMBR hypervisor as described in the MU research of Case #2 (King, Samuel T. 2006) was embedded in the Intel motherboards BMC flash memory, subsequently improved and finally working with another one as »nested hypervisors«. The purpose of such a project is not known to us and, because the hypervisor has been embedded without customer notification, we consider it being malicious.
2. The combination of the MH embedded in BMC flash memory or downloaded in it together with a vulnerability of IPMI (US CERT 2013 and Bonkoski, Anthony J. 2013) can be used to exploit millions of computers worldwide. Thus, the risk exposure is very high.

| Run# | Exec. Time Tci | Deviation Dci |
|---|---|---|
| 1 | 5.5013 | 0.00081 |
| 2 | 5.4992 | 0.00062 |
| 3 | 5.5017 | 0.00093 |
| 4 | 5.4987 | 0.00033 |
| 5 | 5.5024 | 0.00060 |
| 6 | 5.5035 | 0.00135 |
| 7 | 5.5006 | 0.0017 |
| 8 | 5.5002 | 0.00082 |
| 9 | 5.4981 | 0.00057 |
| 10 | 5.4998 | 0.00088 |
| | AvTc=5.5005 | AvDc=0.0008 |

Table 1 – Testing phase without hypervisor

| Run # | Test #1 | | Test #2 | |
|---|---|---|---|---|
| | Exec. Time Thi | Deviation Dhi | Exec. TimeThi | Deviation Dhi |
| 1 | 5.5553 | 0.0069 | **5.6109** | **0.0328** |
| 2 | **5.6209** | **0.0312** | 5.5591 | 0.0034 |
| 3 | 5.5507 | 0.0023 | 5.5641 | 0.0070 |
| 4 | 5.5492 | 0.0082 | **5.5918** | **0.0544** |
| 5 | 5.5651 | 0.0058 | **5.6125** | **0.0255** |
| 6 | 5.5584 | 0.0076 | **5.5774** | **0.0224** |
| 7 | **5.7345** | **0.0190** | 5.5662 | 0.0037 |
| 8 | 5.6233 | 0.0018 | **5.5771** | **0.0277** |
| 9 | **5.6185** | **0.0337** | **5.7033** | **0.0471** |
| 10 | **5.6997** | **0.0423** | **5.5723** | **0.0151** |
| | AvTh1=5.6076 | AvDh1=0.0159 | AvTh2=5.5935 | AvDh2=0.0239 |

Table 2. First two tests (each has ten runs) *with hypervisor*

| Run # | Test #3 | | Test #4 | |
|---|---|---|---|---|
| | Exec. Time Thi | Deviation Dhi | Exec. Time Thi | Deviation Dhi |
| 1 | **5.6352** | **0.0283** | 5.5614 | 0.0042 |
| 2 | **5.6642** | **0.0649** | **5.5652** | **0.0200** |
| 3 | 5.5637 | 0.0039 | 5.5527 | 0.0019 |
| 4 | 5.5438 | 0.0082 | 5.5607 | 0.0060 |
| 5 | 5.5660 | 0.0019 | **5.5862** | **0.0424** |
| 6 | **5.5593** | **0.0189** | **5.5969** | **0.0288** |
| 7 | 5.5572 | 0.0052 | **5.6439** | **0.0420** |
| 8 | **5.7767** | **0.0153** | **5.7073** | **0.0459** |
| 9 | **5.5868** | **0.0399** | **5.6006** | **0.0255** |
| 10 | 5.5516 | 0.0086 | 5.5452 | 0.0010 |
| | AvTh3=5.6005 | AvDh3=0.0195 | AvTh4=5.5920 | AvDh=0.0217 |

Table 3. Additional two tests (each of ten runs) *with hypervisor*

We highlighted in bold runs with a deviation of >= 0.01.
Deviation values fall into two big groups:
Group 1: less than 0.01 – here we have 18 values
Group 2: more than 0.01 – here we have 22 values.
Statistically, it's close to 50% probability to fall into each of those groups.

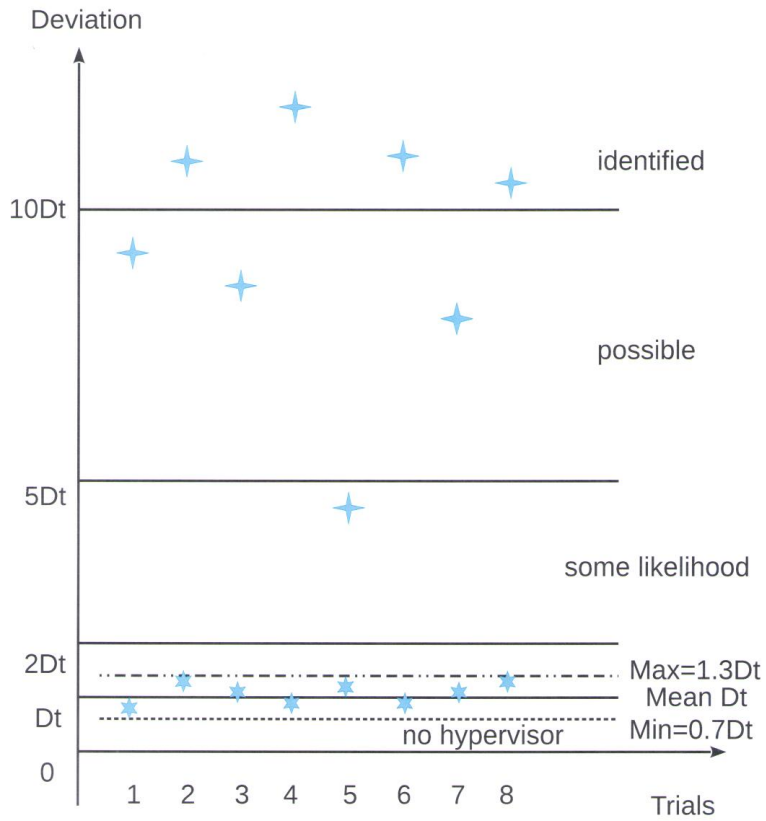


FIG. 1 Deviation distribution in 8 – trials test

- trials without hypervisor

- trials with hypervisor

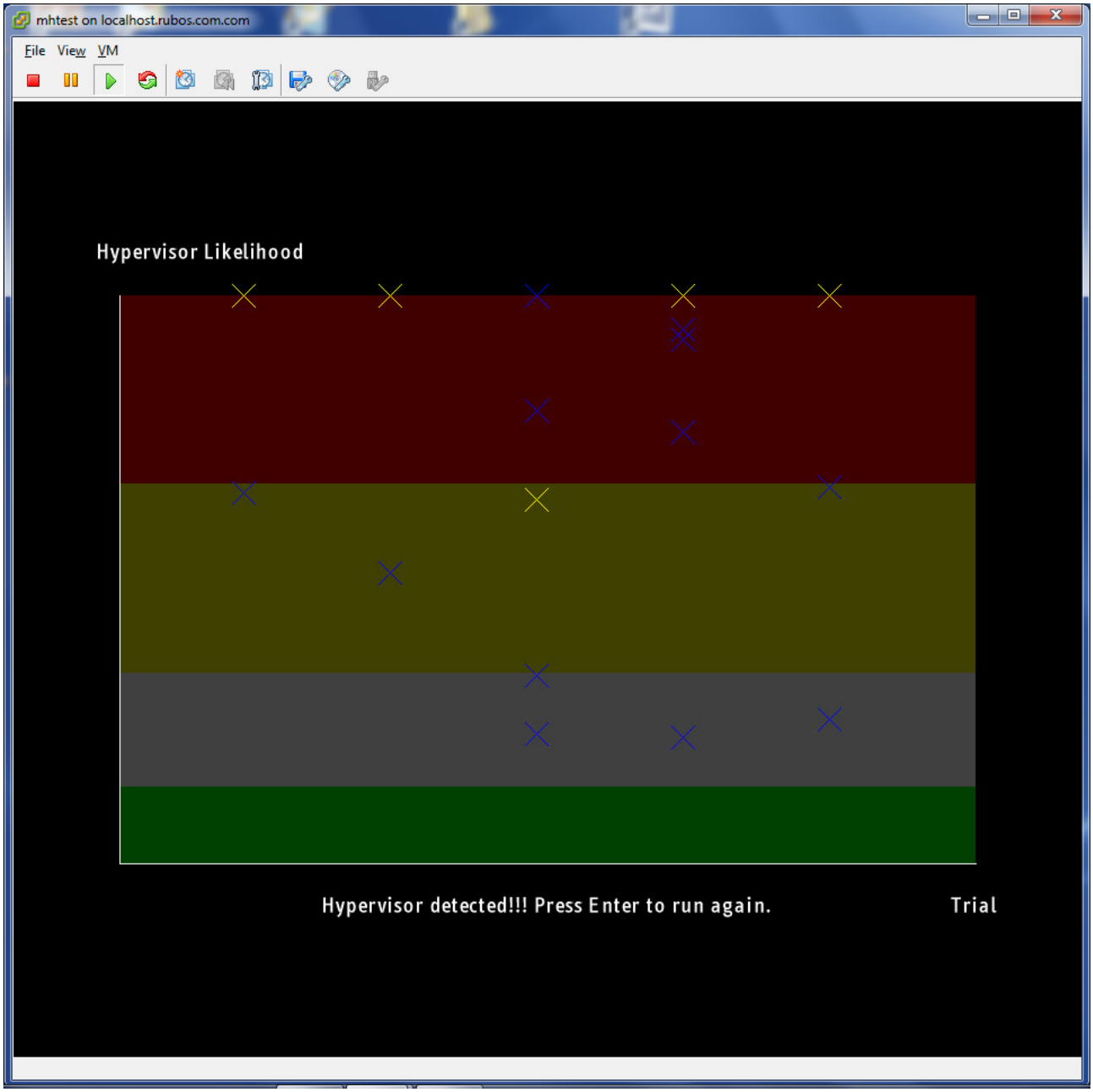


Fig. 2 – The output of HyperCatcher 1.0 software #1

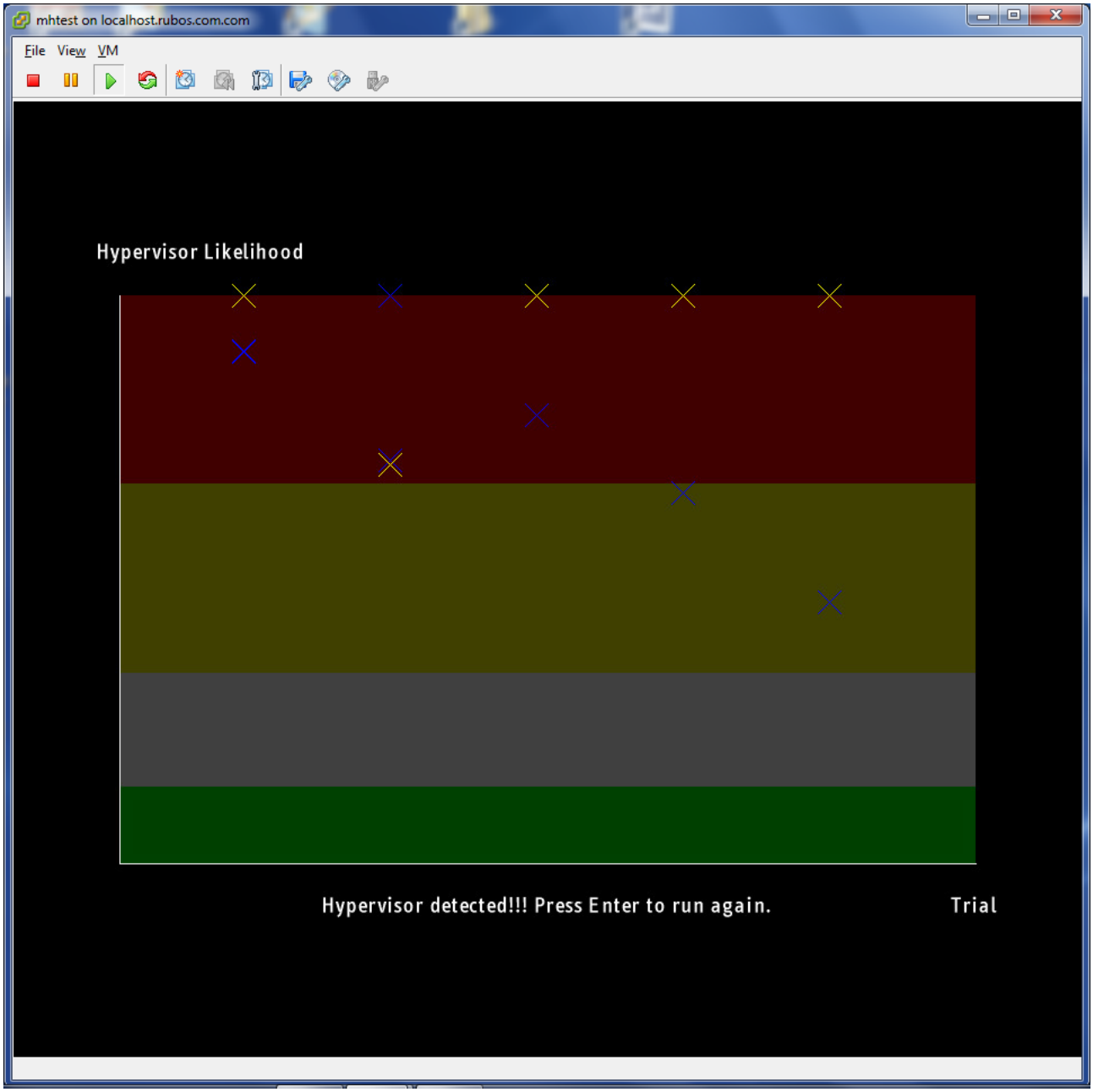


Fig. 3 – The output of HyperCatcher 1.0 software #2

3. We introduced a definition of *computing technology vulnerability* and used this category to explain risks beyond and above bugs or software flaws. It is not easy to convince major IT vendors (in our case the Intel Corporation) of the fact that technology solutions may be vulnerable, thus creating security risks. Therefore we should not expect vendor initiatives to fix *technology vulnerabilities*.

   A mandatory implementation of a Security Development Life Cycle would prevent us from technology vulnerabilities in the future.

4. To address the identification of MH activity we designed and developed a software outside of mainstream MH research. It is efficient and precise, can identify an MH within minutes and with 99.9% accuracy.

   Now we have a first layer of protection: The detection of MHs. Still, it's only one, yet a very important step towards full protection.

5. Our software identification activity cannot be detected by a MH, because we use a general purpose application for this purpose. We changed the execution time method to measure our programs runtime instead of measuring various system and hypervisor related operations. There are two methods of MH detection and each works efficiently and reliably. The HyperCatcher v.1.0 is a first production version utilizing both methods and can run either as boot loaded software or within an installed guest OS. The software does not require any specific skills to use – it identifies a MH automatically.

6. We hope to see more security research in detecting and protecting against the MH threat. We hope that the security community will take the lead, fixing the technology vulnerabilities we discussed.

# 8 About the Author

Mikhail A. Utin, CISSP. PhD completed his basic engineering education in 1975 in Computer Science and Electrical Engineering. His career in Russia included working for several research and engineering organizations. Doctorate / PhD in Computer Science (1988) from then Academy of Science of the USSR. From 1988 to 1990 he founded an information technology company and successfully worked in t he emerging Russia's private sector. He had several USSR patents and published numerous articles. Immigrated in the US with family in 1990 to escape from political turmoil and hoping to continuing his professional career. Worked in the US in information technology and information securit y for numerous companies and organizations including contracting for US government DoN and DoT. Together with colleagues he formed the private company Rubos, Inc. for IT security consulting and research in 1998. The company is a member of ISSAs New England chapter. (ISC)2 certified professional for ten years. Published articles on the Internet and in professional journals, and reviews articles submitted to the (ISC)2 Information Security Journal: A G lobal Perspective. Current research focus on information security governance, regulations and management, and the relationship between regulations, technology, business activities and businesses' security stat us. Most of the research is pioneering work never discussed by the information security community.

This paper has been published on behalf of Rubos Inc. team that made this research possible

# 9 Bibliography


- Utin, M. (2014). A Myth or Reality – BIOS-based Hypervisor Threat (DeepSec 2014). https://deepsec.net/docs/Slides/2014/A_Myth_or_Reality___BIOS-based_Hypervisor_Threat_-_Mikhail_Utin.pdf
- Utin, M. (2016). Malicious Hypervisor Threat – Phase Two: How to Catch the Hypervisor (DeepSec 2016). https://deepsec.net/docs/Slides/2016/Malicious_Hypervisor_Threat_Mikhail_Utin.pdf
- King, S. Cell (novel). (2006). Wikipedia. The free Encyclopedia. On-line; accessed 15-October-2016. Retrieved from https://en.wikipedia.org/wiki/Cell_(novel)
- Xakep.ru (2011). Chinese Add-ons: True Stories of virtualization, information security and computer spying**.** Translated from Russian, Copyright © DeepSec GmbH and Rubos, Inc., 2014.
- Utin, M. (2015). From Misconception to failure – Security and Privacy in US Cloud Computing FedRAMP Program. In S. Schumacher and R. Pfeiffer, R. (Editors). In Depth Security: Proceedings of the DeepSec Conferences (Pages 255-314). Magdeburg: Magdeburger Institut für Sicherheitsforschung.
- Scribed.com. (2016). Intelligent Platform Management Interface Implementer's Guide; Draft-Version 0.7, 9/16/98. [On-line; accessed 15-October-2016]. Retrieved from https://www.scribd.com/document/4026738/Intelligent-Platform-Management-Interface-Implementer-s-Guide-Draft-Version-0-7
- King, Samuel T., Chen Peter M. (University of Michigan),Wang, Yi-Min, Verbowski, Chad, Wang, Helen J., Lorch, Jacob R. (Microsoft Research) (2006). SubVirt: Implementing malware with virtual machines.. IEEE Symposium on Security and Privacy, Berkley/Oakland, CA, USA, 21-24 May, 2006.
- Bonkoski, Anthony J., Bielawski, Russ, Halderman, Alex J. Illuminating the Security Issues Surrounding Lights-Out Server Management . Michigan University. (2013). 7$^{Th}$ USENIX Work-

shop on Offensive Technologies, August 13, 2013, Washington, DC. Retrieved from https://www.usenix.org/conference/woot13/workshop-program/presentation/bonkoski

- US CERT (2013). Alert TA13-207A. The Risks of Using the Intelligent Platform Management Interface (IPMI). US CERT, July 26, 2013. [On-line; accessed 15-October-2016]. Retrieved from https://www.us-cert.gov/ncas/alerts/TA13-207A
- Wikipedia. (2016a) Vulnerability – Wikipedia. The free Encyclopedia. [On-line; accessed 15-October-2016]. Retrieved from https://en.wikipedia.org/wiki/Vulnerability
- Mackie, Kurt (2013). News. Microsoft Profiles Hyper-V Improvements in Windows Server 2012 R2. 10-June-2013. [On-line; accessed 15-October-2016]. Retrieved from https://redmondmag.com/articles/2013/06/10/windows-sever-2012-r2-hypervisor.aspx
- Wojtczuk, Rafal and Joanna Rutkowska (2009). Attacking Intel Trusted Execution Technology. Black Hat DC. February 18 – 19, 2009. DC, USA.
- Wojtczuk, Rafal and Joanna Rutkowska (2011). Attacking Intel TXT via SINIT code execution hijacking. November 2011. [On-line; accessed 15-October-2016]. Retrieved from https://invisiblethingslab.com
- Korkin, Igor (2015). Two Challenges of Stealthy Hypervisor Detection: Time Cheating and Data Fluctuations. CDFSL 2015.