A Methodology for Testing Virtualisation Security

Scott Donaldson, Natalie Coull and David McLuskie
Abertay University, Dundee, Scotland

This is the accepted version of a paper presented at the *International Conference on Cyber Situational Awareness, Data Analytics And Assessment (CyberSA 2017)*, June 19-20, 2017, London, UK which will be published by IEEE

A Methodology for Testing Virtualisation Security

Scott Donaldson, Natalie Coull and David McLuskie
Abertay University, Dundee, Scotland

ABSTRACT

There is a growing interest in virtualisation due to its central role in cloud computing, virtual desktop environments and Green IT. Data centres and cloud computing utilise this technology to run multiple operating systems on one physical server, thus reducing hardware costs. However, vulnerabilities in the hypervisor layer have an impact on any virtual machines running on top, making security an important part of virtualisation.

In this paper, we evaluate the security of virtualisation, including detection and escaping the environment. We present a methodology to investigate if a virtual machine can be detected and further compromised, based upon previous research. Finally, this methodology is used to evaluate the security of virtual machines.

The methods used to evaluate the security include analysis of known vulnerabilities and fuzzing to test the virtual device drivers on three different platforms: VirtualBox, Hyper-V and VMware ESXI. Our results demonstrate that the attack surface of virtualisation is more prone to vulnerabilities than the hypervisor. Comparing our results with previous studies, each platform withstood IOCTL and random fuzzing, demonstrating that the platforms are more robust and secure than previously found.

By building on existing research, the results show that security in the hypervisor has been improved. However, using the proposed methodology in this paper it has been shown that an attacker can easily determine that the machine is a virtual machine, which could be used for further exploitation. Finally, our proposed methodology can be utilised to effectively test the security of a virtualised environment.

Keyword: Networking, Security, Virtualisation

INTRODUCTION

Virtualisation involves isolating the operating system (OS) in a virtual machine (VM) which enables the virtual layer to present hardware to the guest operating systems running on top. However, there are some concerns that the virtual layer adds new attack vectors to a network, which have not been fully researched. The virtual layer is a security risk in that if it were to be compromised, an attacker could potentially gain control of all the systems that run on top of it. According to Gartner (Gartner, 2010) an estimated 60 percent of virtualised servers implemented will be less secure than the physical servers they replace.

Virtualisation has been rapidly adopted by enterprise for consolidation of servers as well as desktop virtualisation for system testing and development. One of the key concepts of virtualisation is that guest operating systems in VMs should be isolated duplicates of real machines, meaning they should behave in the same way. However, security researchers have found that VMs are not completely isolated and that the isolation can be broken. A vulnerable hypervisor could expose the virtual layer to several attacks including allowing arbitrary code from the guest operating system to be run on the host system. This type of attack is known as 'escaping the virtual machine'. If an attacker can execute commands from a guest on the hypervisor or underlying host they would have complete control over the virtual environments that are running on the host machine. Virtualisation security is one of the most important parts of a modern network as it is a critical point of compromise. This paper aims to evaluate the security of virtualisation, including its resilience to detection and to propose a methodology that could be used as part of the penetration testing methodology to aid security researchers in evaluating the security of a virtual environment.

LITERATURE REVIEW

There are many new security challenges within virtualisation. These attacks can range from hyper jacking (Sgandurra & Lupu, 2016), guest jumping (Huang, Chen, Shih, & Lai, 2012), compromised administrator tools (Criscione, 2010) and exploiting software vulnerabilities (Economou, 2011). Although virtualisation is meant to somewhat ease system administrator's workloads there are additional considerations that need to be made such as security policies for VM access, as well as vulnerability analysis and patch management for enterprise networks.

Virtualisation Detection is often included in the attack surface and should be considered somewhat of a threat (Popek & Goldberg, 1974). If an attacker has been able to detect that a machine is virtualised, they can launch attacks specific to the virtualisation layer. This has implications for security researchers as virtualisation is a common tool used in analysing malware. Ideally, users or attackers should not be able to detect that the system is virtualised.

Virtual machines are now being used to analyse malware as it provides an isolated environment in which the malware can be contained. However, to stop security researchers from reverse engineering or analysing malware, malware writers are starting to include VM detection in their malware functionality, often called VM aware malware. Once the malware detects it is in a VM it can change how it behaves, attack the virtualisation layer or just refuse to run as intended and thus thwarting the proper analysis of the malware program.

From a malicious intruder perspective, once virtualisation has been detected an attacker can launch malicious attacks against it such as a denial of service attack. This can cause the VM to unexpectedly exit or force it to be shut down by the Virtual Machine Monitor (VMM). Other attacks can include further penetrating the virtualisation layer by exploiting software bugs in the VMM or hypervisor.

Ferrie (Ferrie, 2007) details attacks on virtual machine emulators and states that the interest in detecting virtualisation is not limited to VM aware malware but

also to detect if malware is utilising virtualisation to run hidden from the host operating system, for example a hypervisor such as a virtual machine based rootkit. Ferrie describes common ways in which virtualisation can be detected.

In theory, a virtual machine should not be able to be detected as the VMM can intercept sensitive instructions such as CPUID, which is used by software to identify the type of processor that is being used. Ferrie suggests that methods can be used to hide the presence of virtualisation such as clearing the CPUID flag which correspond to the hardware assisted VM extensions, however other artefacts are still present which can be used to indicate virtualisation. Common detection methods and tools include RedPill, interrupt descriptor table (IDT) and Scooby Doo. The method presented by Quist and Smith (Quist, Smith, & Computing, 2006) is based around the local descriptor table (LDT). They state that the reason they use this method for detection is because the SIDT instruction based methods (e.g. RedPill) have various problems if a multi core CPUs are being used. This is because the interrupt descriptor table can change significantly when the process runs on multiple cores (Quist et al., 2006). Ferrie (Ferrie, 2007) further discusses this concept stating that VMware makes use of the LDT, which is not otherwise used by Windows in a non-virtualised system and thus if there was a non-zero LDT base then this is a good indicator that a virtualised system was being used.

In the present state virtualisation is detectable, however detection methods can vary depending on the virtualisation architecture and how the vendors VMM handles non privileged instructions, Ferrie (Ferrie, 2007) concludes that there is ultimately nothing we can do about preventing virtualisation detection as the design of the VMM inherently permits interception of non-sensitive instructions, some of which can cause information leakage such as the SIDT instruction. Importantly, all the methods used to detect virtualisation involve the use of non-privileged instructions which are not intercepted by the VMM or hypervisor and therefore these methods cannot be detected and prevented.

Device Input/output

An empirical study into security exposure of hosts of hostile virtualised environments (Ormandy, 2007) identifies that there are two sub systems that are the most complex components and thus more likely to harbour bugs. One of these sub systems is emulated I/O devices. This subsystem was identified because of handling invalid, illegal or non-sensitive I/O activity, essentially meaning that the sub system is prone to errors. I/O is a major part of virtualisation as it allows the guest OS to communicate with hardware or virtualised hardware. This sub system is complex and keeping the VMM or hypervisor small is difficult. The bigger a hypervisor the larger the attack surface is because it is more likely to have bugs. According to Karger and Safford (Karger & Safford, 2008) virtualising I/O has always been a more complex problem then virtualising the CPU. The authors describe three design concerns in providing virtualised access to devices:

1. Are device drivers shared?
2. Are they trusted?
3. Where are they located?

Answers to these depends on the virtualisation architecture, each with its own trade-offs in complexity, security and performance. An example of this is Xen where the I/O is separated from the hypervisor, it uses a privileged guest called a Dom0 partition. All VMs (DOMu's) I/O running on Xen will be redirected though the Dom0 partition which in turn has control of the device. This keeps the Xen hypervisor small, yet according to Karger and Safford (Karger & Safford, 2008) this has security implications. The main reason as to why they claim that this has security implications is because the Dom0 partition is shared between all the virtual machines and this means that pure isolation between the VMs cannot be achieved because if the I/O partition is compromised then it could be possible to compromise all the VMs from this partition.

Research Methods Considered

There are not many methodologies developed specifically for testing and researching virtualisation security however some researchers have applied common testing methods to investigate security in this area. Criscione (Criscione, 2010) developed a virtualisation assessment toolkit (VASTO) for Metasploit which aims to aid in a penetration test of a virtualised system.

A penetration test involves assessing the full security of an infrastructure i.e. escaping and compromising the hypervisor is not the only goal. The aim is to totally own the whole environment. A standard penetration test would not fully consider all the virtual layers and as a result will not fully test the virtualisation attack surface.

One of the reasons as to why the virtualisation layer is not fully tested is because According to Criscione (Criscione, 2010), tools are not virtualisation aware and knowledge of virtualisation security issues are not well known. However, the VASTO toolkit for Metasploit aims to solve this problem by being one of the few packages specifically created for testing virtualisation security.

Ormandy (Ormandy, 2007) uses a combination of methods to find vulnerabilities in different virtualisation vendor software. Ormandy uses a combination of manual analysis of code where source was available and closed source software analysis, which involves black box testing such as fuzzing.

Economou and Horan (Economou, 2011) used a similar method to investigate virtualisation security. Economou took interest in a security patch bulletin posted by Microsoft MS10-102 Hyper-V VMbus vulnerability CVE-2010-3960. The exploit developed allows a Denial of Service (DoS) attack on the Hyper-V service, this is done from a guest OS and crashes the main service affecting all VM running on the hypervisor. To further investigate the vulnerability the authors started by trying to find the original bug by reversing the patch. This was done by installing the patch and looking at what files were changed. The files modified were vmbus.sys, vmswitch.sys and storvsp.sys. After running a binary difference tool, changes were found to a function's name across the three files, according to Economou and Horan the change made was to split one function into two.

Each method for investigating virtualisation presented here highlights that current methods used for investigating security such as penetration testing methodologies and stress testing can be applied to virtualisation if modified. The main aim of this paper is to propose a methodology that can be used to help a security researcher perform a penetration test to evaluate the security of virtualisation systems. Once the methodology has been proposed the next aim of this paper is to evaluate the effectiveness of the methodology for use in the penetration testing process.

PROCEDURE

This section describes our procedure for evaluating the security of virtualisation. This includes virtualisation detection and escaping the environment.

This can be done using our proposed methodology to assess different types of virtualisation architectures to identify vulnerabilities. Previously discovered vulnerabilities can also be used to demonstrate the potential risk of security issues that are possible in a virtualised environment. The methodology is based on the procedure used by security researchers to find vulnerabilities in virtualisation. Three approaches are tested:

- Penetration testing methodology presented by Criscione (Criscione, 2010) using VASTO an exploit package for Metasploit.
- Stress testing (fuzzing) with further analysis by Ormandy (Ormandy, 2007).
- Reverse patch investigation and debugging by Economou and Horan (Economou, 2011).

Methodology for investigating virtualisation security

The methodology that will be adopted for this paper is as follows:

I. Discovery
    a. Detecting Virtualisation
II. Information gathering
    a. Identifying virtualisation vendor or type
    b. Identifying services and versions running
III. Vulnerabilities and exploits
    a. Demonstrating flaws in virtualisation found from CVE investigation
IV. Fuzzing and further investigation
    a. Apply fuzzing techniques to I/O
    b. Demonstrating any flaws found from fuzzing

The first stage of the methodology is Discovery; this is used to identify if the system is virtualised. Tools such as Metasploit have built-in scripts that can be utilized in our procedure to test and discover virtualisation.

Information Gathering and Enumeration is another important part of penetration testing. This stage gathers more information about the system, such as operating system, version of operating system, patches installed and virtualisation software being used. The information is then used in the next stage of the methodology, Vulnerabilities and Exploits. This can be used to analyse the

effectiveness of tools like VASTO and others at gathering and enumerating information from virtualised environments.

From the information gathered, exploits can be used on any vulnerabilities found. This stage of the methodology involves identifying known exploits and vulnerabilities. Analysis of previous exploits and vulnerabilities will aid in the fuzzing stage to identify new, unknown vulnerabilities. The practical work here will be carried out using VASTO as a penetration testing tool for exploitation.

The results gathered from the methodology will be used as evidence to address the aim of the project. Analysis and discussion of the results will be used to evaluate how secure virtualisation security is.

Lab Configuration

Table 1 shows the configuration of the test labs used for the methodology and experiments. Each virtualisation platform was configured to use bridged networking and Intel virtualisation technology extensions (VT-x).

Platform	Operating System
VirtualBox	Windows 7
Windows Server 2008 R2 (Hyper-V)	Windows 7
VMWare ESXI 5.0.0	Windows 7

TABLE I. TEST CONFIGURATIONS

Discovery

There are several methods which can be used to detect virtualisation using low level instructions. Here, we *use* a Metasploit Post Exploitation script to detect virtualisation on multiple virtualisation platforms and analyse their effectiveness.

1) Metasploit Checkvm Post Exploitation

During a penetration test it may not be possible for the tester to determine if a system is running in a virtualised environment remotely. However, once they have compromised the system, tools can be used locally to detect virtualisation in the post exploitation phase. The Meterpreter is Metasploit's post exploitation module and boasts various tools which can be used to further compromise a network or maintain access. There are three scripts that can be used relating to virtualisation however only one for detection, checkvm. The checkvm script is used on all the virtualisation platforms tested in the methodology.

Firstly, to get the Meterpreter connected the guest machine needs to be compromised with the Meterpreter payload, this can be done using various methods. Once the guest OS is compromised the penetration test moves to the post exploitation phase. A hacker may want to detect if the machine that has been compromised is virtualised. To test for this checkvm is used in the Meterpreter shell.

2) Virtualisation artefacts in the guest OS

Guest additions are likely to be installed on the guest OS to support better performance, shared folders and additional tools. The guest additions will install as a regular program in Windows creating registry entries, install folders, files and services. Checking the running processes and services should reveal information. Guest additions are likely to be installed in a virtualised guest OS as there is a big performance and stability gain when they are used. The guest tools will leave footprints behind on the guest OS and these artefacts can be used during a penetration test to reveal that a virtualised system is being used. A process was developed to analyse the footprints. Guest additions were installed on all guest OS's, each platform has their own method of installing these tools. Hyper-V will install the guest additions and drivers automatically after the OS is installed using plug and play. VMware and VirtualBox both have an .ISO file that is used and mounted inside the VM, from there the tools are installed as normal applications.

The following process was used on all three platforms to check for virtualisation artefacts:

1) Checking for installation files and folders
2) Checking for running processes and services

Information Gathering

The information gathering phase of a penetration test involves enumerating services to gather more information about the system. This phase of the methodology attempts to try and identify which versions of each virtualisation platform tested is running, as well as gathering more information about the virtualised environment.

1) Version Information

Post exploitation techniques can be used locally to gather information about the system such as what device drivers are installed. Running Metasploit's enum_devices script can provide information about device drivers to identify the underlying operating system version.

Vulnerabilities and Exploits

Analysis of previous exploits and vulnerabilities will provide more information about where common vulnerabilities are in virtualisation and how they are exploited. The analysis of previous vulnerabilities will also provide information for the next phase of the methodology which is intended to find similar vulnerabilities, as discussed in the literature review.

1) VASTO Install

VASTO is an exploit pack developed by Criscione (Criscione, 2010). This tool will be used to attack the surface of virtualisation demonstrating that the attack surface provides ways in which to penetrate virtualisation other than directly exploiting the hypervisor.

2) Client Side MITM attack (VMWARE_vilurker exploit)

VMware has a binary client for accessing and managing the virtualisation infrastructure called vSphere. The attack to be analysed is a MITM on the

client, all versions of VMware are susceptible to this type of attack. The attack exploits the auto update feature. An XML file is sent in the handshaking process which determines what version of the client (vSphere) should be running. If it is not running the correct version, then it is directed to a URL which will link to the installer.

This opens the handshaking process to MITM attacks, a crafted XML file can be sent with a modified version of vSphere. Criscione (Criscione, 2010) demonstrated this attack on VMware ESXI 4.0.0 using VASTO. This tool will be used to analyse how successful it is on the newer version of VMware ESXI 5.0.0.

a) Exploit Configuration

The module used in this attack is VMware_vilurker. This module will run a server holding the modified vilurker.exe to be run on the client machine.

b) Client side MITM

On the client side a MITM attack can be used to get the clients vSphere application to access the server now running on backtrack. For simpler analysis, the vSphere application was pointed to backtrack using the IP address to directly connect.

When the client application connects to the MITM server the user will be presented with an SSL warning dialogue box and the user will be asked if they want to proceed. If this is accepted the user will be prompted to download the malicious installer which they will think is just an update for the application. If they click no then the process can be repeated later. The user can either run the update straight away or save it for later.

Once the update has been installed the payload can then be loaded on to the client machine and executed. Because VASTO uses Metasploit there are several payloads that can be used with this attack.

Virtual Device Driver Fuzzing

All virtual devices in a virtual machine are emulated in some way, the emulation depends on the virtualisation platform. From the literature reviewed, the area to focus on for virtualisation security is I/O device emulation. Ormandy (Ormandy, 2007) uses stress testing (fuzzing) to analyse device I/O for errors, once an error was found it was investigated further to see if it was exploitable.

This phase of the methodology aims to build on the research already done and further the knowledge of virtualisation security. The results of which will determine if security in device I/O has improved since the time of the research discussed in the literature review.

The method used in this phase involves testing input/output controls (IOCTL) specifically in Windows guest OS's. IOCTL is a gateway for user mode applications to access kernel functions, for example accessing devices. Hardware devices are only addressable from the kernel, if applications running in user space want to communicate with the underlying hardware a device

driver is used. An example of a typical vulnerability in a device driver is where a local user can pass invalid buffers to IOCTL calls and cause a system failure similar to (CVE-2011-2305) analysed later in this section.

The experiments will work by sending random data via IOCTL to the driver which will in turn communicate with the kernel to make a system call to the device. Because the guest OS is virtualised the emulated device will be queried with the data passed to it. If there is a flaw in the emulated device it will cause an error, most likely crashing the hypervisor or creating some sort of warning.

1) Virtual Device Driver Fuzzing Experiments

a) IOCTLFuzzer

At the start of this section two vulnerabilities in virtual device drivers were analysed. Both CVE-2011-2305 and CVE-2007-5671 were found to *have* vulnerabilities in the guest driver which is used for communication between the guest and host. The device driver is what allows communication between the guest OS and host or hypervisor. From the literature reviewed, the future research stated by various authors suggests that device I/O fuzzing should be the focus of security research.

This phase of the methodology will analyse IOCTLs using IOCTLfuzzer to analyse and evaluate fuzzing as a method of efficiently testing virtual device I/O security.

Attempts by the guest operating system to access the hardware are routed to the virtual device driver using IOCTLs which in turn interact with the virtualisation I/O stack. Testing of this area seeks to exercise the underlying virtualisation layer and how it handles these requests. IOCTLfuzzer uses a driver that hooks into DeviceIoControlFile to manipulate and fuzz IOCTLs to the system. The procedure outlined below will be used on all three platforms.

b) Attack Surface Analysis

To get the maximum out of the testing, the drivers attack surface needs to be analysed, this is done using the attack surface analysing feature, as shown below:

1. ioctlfuzzer.exe –boot

The next time the system restarts ioctlfuzzer will log all the information found from IOCTL requests. Analysis of the log file will show all the collected IOCTL information which can be used for fuzzing. The analysis is done with the command shown below:

2. ioctlfuzzer.exe --analyze --loadlog %SystemDrive%\ioctls.log

c) IoSpy and IoAttack

IoSpy and IoAttack are two tools that are part of the Windows driver development environment used for testing IOCTLs and WMI requests. IoSpy is

used to analyse the environment it does this by monitoring IOCTL requests and logging them. IoAttack then uses this information from IoSpy to target specific drivers. This phase of the methodology will analyse IOCTLs using IoSpy and IoAttack to analyse and evaluate fuzzing as a method of efficiently testing virtual device I/O security.

RESULTS

In this section, the results from the experiments are presented in the same order in which the methodology was carried out.

Discovery Analysis

The methods used in this section show that virtualisation detection is possible. For each of the three platforms tested, the virtualised operating system revealed information which demonstrated that they were in a virtualised environment. This can have an impact on how malware operates and poses a significant threat to virtual honeypots and how they operate, since it means that malware writers can detect the VM and stop researchers from finding out how the malware works.

1) Detection using checkvm Metasploit

Table II shows the results for the checkvm script using the post exploitation module available in Meterpreter.

Out of the three platforms tested using this method all were detected correctly using the checkvm Meterpreter script. This information reveals that the guest OS is running on a virtualisation platform.

Platform	Virtualisation Detected?
VirtualBox	Yes
Windows Server 2008 R2 (Hyper-V)	Yes
VMWare ESXI 5.0.0	Yes

TABLE II. RESULTS OF VIRTUAL MACHINE DETECTION

2) Virtualisation artefact findings

All the platforms were tested for software artefacts after the guest additions were installed on the virtual machine. The results as shown in Table III found that all guest addition tools leave behind artefacts that can be used to identify the virtualisation platform. Artefacts found included installation files, system services and running processes.

Platform	Artefacts Detected?
VirtualBox	Yes
Windows Server 2008 R2 (Hyper-V)	Yes
VMWare ESXI 5.0.0	Yes

TABLE III. RESULTS OF ARTEFACT DETECTION

Information Gathering

1) Version Information

Metasploit's enum_devices script successfully revealed all drivers running for that particular operating system. For each virtual machine tested, driver names were returned which indicated that the underlying architecture was a virtual machine, for example "VMWare Tool Service" and "VirtualBox Device". This information can be used to fingerprint the underlying virtual machine software. All platforms tested returned virtualised devices that can be used to detect virtualisation. The information gathered could be useful if there is a known vulnerability in a driver that allows privilege escalation in the guest OS.

Previous Vulnerabilities and Exploits

1) Client side MITM attack findings

The results from the client side MITM attack show that virtualisation adds more risk to a network as it can be used to compromise machines. Figure 1 shows what the client sees after the malicious installer has been *run,* it notifies the user to restart the application

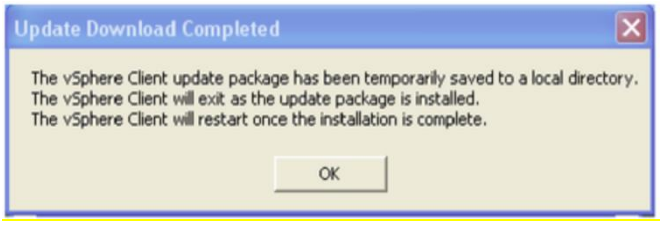

FIGURE 1 CLIENTS VIEW AFTER COMPROMISE

After the malicious client is installed the payload is run and executed, bind TCP was used in the experiment. The server will receive a message that the client has connected and downloaded the executable. Using Metasploit a bind TCP handler can be started which connects to the client machine which is now compromised.

The results from this experiment show that this type of attack is still possible using version 5.0.0 of VMware ESXI. However, the attack is only made possible by the user accepting the SSL certificate. Furthermore, it shows that virtualisations attack surface adds an increase risk to the network by providing another attack vector.

Virtual Device Driver Fuzzing Findings

1) Crashme Results

Crashme was executed on all three platforms to stress test the robustness of the underlying virtualisation layers. The program was run for 10 hours on each platform with logging enabled and verbose mode set to 3. Once the program had finished, the logging information was analysed for anomalies.

There were no significant findings in the results from this experiment even after being subject to 10 hours of stress testing. All three platforms were unaffected by Crashme. When compared with the findings of Ormandy

(Ormandy, 2007), the results show that virtualisation security has improved vastly in this area from the time of Ormandy's initial research.

2) IOCTLfuzzer Findings

a) Attack surface results

The results from the attack surface show that drivers were found that could be fuzzed using IOCTLs. All the platforms tested returned similar results to that of the information gathered at the start of the methodology. The information gathered in this stage is used in the next phase when fuzzing DWORDS with random data. The reconnaissance and attack surface built up a picture of what devices were available to be fuzzed. This was successful on all platforms tested.

No meaningfully results were obtained from this process on any of the three platforms tested. Like that of the Crashme results, this indicates that the security in this area has been improved.

3) IoSpy and IoAttack Findings

IoSpy also provide a similar attack surface analysed to IOCTLfuzzer, however this was required to be run before IoAttack would run. This was so that the tool could capture IOCTLs to be used in the fuzz testing. Only two drivers were found to be fuzzable by IoSpy using the VirtualBox platform. The procedure was also repeated on Hyper-V and VMware with results returned showing drivers that were fuzzable.

Because the results of IoSpy were limited on all platforms, the test case lists were small. From analysis of fuzzing using these tools on all three platforms, no results were obtained that pointed to bugs or vulnerabilities in the drivers tested.

Our results demonstrate that it is relatively easy to identify the presence of a virtualised environment due to how device drivers are used and reported in the environment. Information gathering and enumeration reveal more information about virtualisation, specifically the Metasploit modules that reveal virtual machines in ESXI's data store. These results have a significant impact on virtualisation security. The findings explored in the MITM attack show that the attack surface of virtualisation needs to be moderated.

Furthermore, the results from fuzzing show that fault tolerance and isolation have improved and that resources are relatively security to IOCTL fuzzing. The analysis of the results from testing virtual device drivers shows that security improvements have been made.

DISCUSSION

Virtualisation security has an impact on malware research, virtual honeypots and cloud computing as well as any other implementation that uses virtualisation. Security issues in virtualisation are therefore high risk and security should be considered a high priority. Existing research provides an analysis of virtualisation and an evaluation of the attack surface which was utilized for our methodology.

Our literature review highlighted the complexity and wide attack surface of virtualisation, and we have only been able to address a subset of this attack

surface in this paper. This was done without compromising the results that could affect a full analysis of virtualisation security. Due to hardware limitations Intel VT-x could not be analysed. This meant that pass-through functionality of virtualisation platforms was not analysed as part of the methodology.

The results revealed that all the operating systems tested contained artefacts relating to virtualisation as shown in tables II and III. From the results, it shows that hiding the presence of virtualisation is extremely challenging if not impossible as artefacts are left behind by each platform tested. Virtualisation can be detected using low level instructions or by looking for artefacts such as device driver information.

Detecting virtualisation can be a threat however this depends on the situation. Because most servers are being or are already virtualised, detection is not as big of a threat as systems are more likely to be virtualised. If malware detects virtualisation and does not run then malware becomes ineffective, assuming the virtualised environment is the end destination. Virtual honeypots can become more effective as systems are more likely to be in a virtual machine, therefore attackers are unlikely to leave if they detect virtualisation. However, from the findings of the enumeration and information gathering experiments further probing after detection reveals information about the environment which could be used against it

The results from enumeration and information gathering show that systems can reveal device version information. Attackers can then look for vulnerabilities present in these versions to further compromise the environment. The methodology primarily focused on analysing modules present in Metasploit to demonstrate information that could be enumerated from the ESXI server. The results from these experiments showed that VMware ESXI 5.0.0 can be enumerated to reveal users and groups, user's permissions, all the virtual machines installed on the server and the physical hardware on which it runs. Enumerating what virtual machines are present on the server poses the most risk to a network. The results also reveal the configuration of each virtual machine, including what guest OS is running inside each VM. This can give vital information to an attacker, however the majority of the modules used require logon credentials for the ESXI server.

The main findings of the methodology highlight that the attack surface of virtualisation is more prone to vulnerabilities than the actual hypervisor itself. All the additional components of virtualisation each add to the attack surface. All three platforms tested have very different attack surfaces, however do share similarities in that guest and management tools are used.

The MITM attack demonstrated against the VMware's vSphere client shows that additional risks are present from the addition of virtualisation to the network. A new attack vector is now possible because of the new infrastructure and components added. The MITM attack resulted in the end client machine being compromised by accepting a malicious update to the application. The result from this experiment shows that this type of attack is still possible using the latest version of VMware and should work for any other version. However, the attack is only made possible by the user accepting the SSL certificate. Mitigation for the type of attack demonstrated is using signed SSL certificates and user education.

The Hyper-V DoS attack could not be analysed as the proof of concept source code returned several warnings and errors. Attempts were made to correct the issues, none of which worked. However Economou and Horan (Economou, 2011) provide a small description of what is meant to happen. When a value is passed from the guest to the host it is checked against another value, this results in a bug triggering a process which never ends because a flag is not set. This type of attack demonstrates that although the virtual machines may be isolated they can still affect other virtual machines running on the same hardware. This type of attack is amplified in virtual desktop infrastructures and data centres where multiple server will run on the same hardware using virtualisation.

From analysis, interaction between the VM and hypervisor is where the majority of arbitrary code execution or VM escape vulnerabilities are found. The most common two are when isolation between the guest and host is broken or when crafted code from the guest triggers a bug in the way that the hypervisor handles the data passed to it.

While there are other ways in which to compromise virtualisation, the hypervisor remains the main target. From analysis of the literature and experiment results vulnerabilities present in the hypervisor have the biggest consequences that are amplified to all the upper layers.

From the literature reviewed, virtual device drivers were the most talked about security issue that needed further exploration and research. Device drivers were analysed in detail for vulnerabilities and security flaws by way of fuzzing IOCTLs. Three tools were used in this process IOCTLfuzzer, IoSpy and IoAttack neither of which found any security issues in the device drivers tested. This shows that since the time of the research carried out by Ormandy (Ferrie) and others security in device I/O have improved significantly.

Hypervisor security has improved as platforms now have a much smaller footprint, the thin smaller code base means that there is not as big of a level of exposure as previous versions. Countermeasures to most problems found in the hypervisor is to patch it vigorously. Any shared folders or applications used to pass data to one another should be disabled, and a good password policy put in place will fend off any brute force attempts on the ESXI server.

While identifying a critical vulnerability that could enable escaping a virtual machine can be patched, this wouldn’t prevent unknown vulnerabilities from compromising security.

CONCLUSIONS

Virtualisation is a major part of modern networks therefore researching virtualisation security has important benefits. As explored in the attack surface, virtualisation can add additional security risks to the network.

Attacks are amplified in virtual desktop infrastructures and data centres where multiple servers will run on the same hardware using virtualisation. The results of this research demonstrate that although the virtual machines may be isolated they can still affect other virtual machines running on the same hardware. From

analysis, interaction between the VM and hypervisor is where the majority of arbitrary code execution or VM escape vulnerabilities are found.

Each virtualisation platform has a unique attack surface with some similarities, therefore there is a need for a standard methodology or framework that can be adapted to test a unique environment.
Our proposed methodology, based on techniques employed by more traditional penetration testing methodologies demonstrates that this approach can be adopted to test the security of a virtualised environment. The steps in our methodology are:

I. Discovery (i.e. detecting if we are in a virtualised environment)

II. Information gathering (i.e. identifying VM vendor, version and services running)

III. Vulnerabilities and exploits (i.e. demonstrating flaws in virtualisation from previous exploits)

IV. Fuzzing and further investigation (i.e. Applying fuzzing techniques to I/O and finding flaws from fuzzing)

By building on existing research, our proposed methodology has been proven to be viable for use by penetration testers and network managers to more fully test and secure their virtualisation environments.

REFERENCES

Criscione, C. (2010). *Virtually Pwned Pentesting Virtualization.* Paper presented at the BlackHat Conference, USA.

Economou, A. H., A. (2011). Behind the Curtain: A Journey into Reversing the Hyper-V VMBus Exploit (MS10-102). Retrieved from https://www.coresecurity.com/blog/behind-the-curtain-a-journey-into-reversing-the-hyper-v-vmbus-exploit-ms10-102

Ferrie, P. (2007). Attacks on more virtual machine emulators. Retrieved from www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf

Gartner. (2010). Percent of Virtualized Servers Will Be Less Secure Than the Physical Servers They Replace Through 2012. Retrieved from http://www.gartner.com/newsroom/id/1322414

Huang, Y.-L., Chen, B., Shih, M.-W., & Lai, C.-Y. (2012). *Security impacts of virtualization on a network testbed.* Paper presented at the Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on.

Karger, P. A., & Safford, D. R. (2008). I/O for virtual machine monitors: Security and performance issues. *IEEE Security & Privacy, 5*(6), 16-23.

Ormandy, T. (2007). An empirical study into the security exposure to hosts of hostile virtualized environments: Citeseer.

Popek, G. J., & Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. *Communications of the ACM, 17*(7), 412-421.

Quist, D., Smith, V., & Computing, O. (2006). Detecting the presence of virtual machines using the local data table. *Offensive Computing*.

Sgandurra, D., & Lupu, E. (2016). Evolution of attacks, threat models, and solutions for virtualized systems. *ACM Computing Surveys (CSUR), 48*(3), 46.