On the Effectiveness of Virtualisation Assisted View Comparison for Rootkit Detection

Toby J. Richer[1] **Grant Neale**[1] **Grant Osborne**[1]

[1]Defence Science and Technology Organisation
PO Box 1500, Edinburgh, South Australia 5111,
Email: toby.richer@dsto.defence.gov.au

Abstract

There is growing interest in tools for monitoring virtualisation infrastructure and detecting malware within Virtual Machines (VMs). View comparison, or cross-view validation, is a technique for detecting object hiding by malware. It involves comparing different views of system objects to find discrepancies that might indicate the use of object hiding techniques.

We present Linebacker, a system for performing view comparison on VMware vSphere VMs. Linebacker compares external (i.e. hypervisor level) and internal (i.e. guest operating system level) views of process, file and registry objects within VMs to detect rootkits that cloak such objects from the view of the guest operating system. We use Linebacker to compare the efficacy of the view comparison technique to sandboxing or API call monitoring approaches to rootkit detection. We also present a case study evaluating the performance impacts associated with using Linebacker to monitor VMs in a production environment. We present execution and analysis time metrics for this study and discuss feedback provided by users.

Finally, we analyse our results and make recommendations regarding the implementation of view comparison for real-world virtualisation infrastructure.

1 Introduction

There is growing interest in tools for monitoring virtualisation infrastructure and detecting malware within Virtual Machines(VMs).

Rootkits often use evasion techniques to avoid detection and thus increase their chance of persisting on a system. Evasion techniques may include manipulation of the way a system reports active processes, lists files and their contents, and displays registry keys. View comparison (Wang, Vo, Roussev, Verbowski & Johnson 2004) is a technique for detecting rootkits by comparing internal and external views of operating system objects. The ability of the technique to detect rootkits has been demonstrated previously (Garfinkel & Rosenblum 2003, Wang, Beck, Vo, Roussev & Verbowski 2005, Quynh & Takefuji 2007, Jiang, Wang & Xu 2007, Jones, Arpaci-Dusseau & Arpaci-Dusseau 2008, Wang, Hu & Li 2011), but to our knowledge it has not yet been implemented within commercial monitoring tools (Adventium Labs 2014, Trend Micro 2014).

There are potentially several reasons why view comparison is not currently used in commercial monitoring tools. View comparison may not scale well to monitoring a large number of machines. The results of view comparison may be difficult to interpret, and may result in a high false positive rate when applied to production systems. Many current virtualisation assisted implementations of view comparison rely on modifications to existing commercial or open-source hypervisors (Garfinkel & Rosenblum 2003, Quynh & Takefuji 2007, Jiang et al. 2007, Wang et al. 2011, Jones et al. 2008), and it may be that adding this capability to commercial tools is too onerous. It may also be that the same or better performance (in terms of VM impact and detection of rootkits) can be achieved through the commercially available tools that use vShield Endpoint (VMware 2014*d*) such as Bitdefender SVE (Bitdefender 2014), Sophos Antivirus for vShield (Sophos 2014) or Trend Micro Deep Security Antivirus (Trend Micro 2014).

We have developed a system for performing view comparison and live Virtual Machine Introspection (VMI)-based (Garfinkel & Rosenblum 2003) forensics on VMware vSphere VMs. Our system supports the "vanilla" VMware ESXi hypervisor and does not require additional drivers (other than VMware Tools) to be installed within monitored VMs. As a result, we have been able to evaluate the performance of our tools in a production environment and gather feedback on the user experience. We also used the Cuckoo malware analysis tool (Cuckoo Sandbox Developers 2014) to sandbox the execution of a sample set of common malware. This sandboxing process was used to generate profiles of the malware, which we used to compare the ability of API monitoring and view comparison to detect rootkits.

This work contributes an analysis of the effectiveness of view comparison detection techniques in detecting object hiding performed by rootkits, as compared to API call monitoring techniques. We assume that the data generated by Cuckoo is similar to the data used by API call-based detection techniques in commercial rootkit detection tools. This data includes windows kernel level activity associated with creation of processes and files, and any subsequent calls made to install filter drivers to hide them from the operating system. To our knowledge, our work is the first analysis of view comparison that compares it to another method of detecting object hiding, over a range of rootkits. Our work also contributes an analysis of view comparison performance across multiple VMs hosted on standard VM infrastructure. Our results provide new insights into the impact of implementing view comparison techniques in real-world

virtualisation infrastructure.

The next section describes previous work in view comparison and introduces Linebacker, our implementation of view comparison for the VMware vSphere virtualisation platform. Linebacker implements view comparison using standard APIs developed by VMware. Section 3 presents an evaluation of the efficacy of performing virtualisation assisted view comparison, based on tests of Linebacker on a corpus of real-world malware samples. Section 4 presents a case study on the performance of Linebacker in a real-world scenario. Finally, Section 5 presents our conclusions and discusses potential further work in this area.

2 View Comparison

View comparison, also known as cross-view validation (Wang et al. 2004, Wang et al. 2005), is a technique for detecting rootkits. It involves comparing different views of system objects to find discrepancies that might indicate the use of evasion techniques by rootkits.

A rootkit is malicious software that allows an attacker to establish and maintain a persistent presence on a machine, while concealing their presence from legitimate users of the machine. Rootkits often hide objects such as files, registry entries, processes and kernel modules in order to avoid detection.

A simple technique for hiding files is to set the hidden bit for each of these files within the Master File Table (MFT). However, the details of such files are still available from the MFT, or by changing system settings to show hidden files.

A more sophisticated hiding technique employed by rootkits is to hook the system Application Programming Interface (API) functions used to enumerate a particular type of object and filter out any results referring to the object being hidden. The underlying data structures storing hidden objects are not modified by this filtering technique. These hidden objects can be detected by comparing the raw contents of these structures to the view presented by the system APIs. Objects that appear within the underlying data structures, but are not visible in API results might have been hidden by a rootkit.

Direct Kernel Object Manipulation (DKOM) rootkits hide objects by modifying data structures in the kernel directly. These rootkits can potentially be detected by comparing API results with raw data structure contents. In some cases they can be detected by comparing the contents of related kernel data structures storing similar information(Wang et al. 2005).

More recent rootkits have implemented new techniques to avoid being detected by view comparison (Kapoor & Mathur 2011). The Koutodoor rootkit blocks read/write access to files, rather than hiding them. The TDSS family of rootkits store files as orphan files in unused portions of the drive. They alter the Master Boot Record (MBR) to allow access to these files without them appearing within the filesystem. Another approach, proposed by Jiang et al(Jiang, Wang & Xu 2010), is for rootkits to modify the fundamental structure of disk or memory to hide processes or files. This would also have the potential to hide processes from Linebacker, since it relies on the reconstruction of disk or memory by VMWare or forensics tools. However, it would be extremely difficult to modify these fundamental structures while enabling the normal operation of other software on the guest OS. Hiding information outside the file system, as done by the TDSS family of rootkits, achieves a similar goal more easily.

Some rootkits do not attempt to hide the presence of the files they have added or altered. One technique used by rootkits is to Hide in Plain Sight (HIPS), by hiding objects within existing files or memory structures or in obscure sections of the filesystem. Rootkits that do not try to hide file objects will not be detected by view comparison, but will potentially be vulnerable to other established rootkit detection techniques.

Virtualised infrastructure provides a significant advantage over non-virtualised infrastructure for the application of view comparison. VMI allows VMs to be observed from the hypervisor, providing an external viewpoint that cannot be tampered with from within monitored VMs. Guest operating system data structures in memory or on disk can be parsed directly from this external view. View comparison can be performed by comparing this external view with the internal view provided by the guest operating system.

2.1 Virtual Machine Introspection

Virtualisation allows multiple VMs to run on a single physical server. Software called the hypervisor, or Virtual Machine Monitor (VMM), presents a virtual hardware platform to VMs and manages accesses to physical hardware. This allows isolation to be maintained between VMs sharing the same physical hardware.

Virtual Machine Introspection (VMI) (Garfinkel & Rosenblum 2003) is the observation of software running inside of VMs from the hypervisor level. The hypervisor controls all access to the virtual hardware platform and the underlying physical hardware, allowing disk, memory, CPU, network and other hardware contents or state to be observed directly, without relying on the operating system running within the VM. Such an operating system is commonly referred to as the guest operating system. VMI has previously been used in a number of applications including Intrusion Detection Systems(IDSs) (Garfinkel & Rosenblum 2003, Jiang et al. 2007, Jones et al. 2008), malware detection and analysis (Jiang et al. 2007, Litty, Lagar-Cavilla & Lie 2008), and forensics (Hay & Nance 2008, King & Chen 2005, Krishnan, Snow & Monrose 2010, Nance, Hay & Bishop 2009).

The key obstacle to VMI is the *semantic gap* between the state of a guest operating system and the raw data visible to the hypervisor. The challenge of reconstructing guest operating system semantics from this external view is well known and a number of methods have been proposed (Garfinkel & Rosenblum 2003, Jiang et al. 2007, Krishnan et al. 2010, Litty et al. 2008, Pfoh, Schneider & Eckert 2009). Pfoh et al. (Pfoh et al. 2009) provide a useful categorisation of such methods into three "view generation patterns": in-band delivery, out-of-band delivery and derivative. These three patterns or combinations thereof encapsulate all possible methods of bridging the semantic gap.

In-band delivery (Pfoh et al. 2009) uses an agent within the VM to deliver information to the hypervisor. This method avoids the semantic-gap challenge by leveraging the guest operating system's inherent knowledge of its own architecture. This method lacks portability and is easily subverted.

Out-of-band delivery (Pfoh et al. 2009) generates an OS-level view from hypervisor-level state information using semantic knowledge obtained in advance. This knowledge might include kernel symbol tables, address space layouts or file system specifications.

The significant body of computer forensics research focused on reconstructing system state from forensic disk and memory images can be applied to overcome the semantic gap when using out-of-band delivery. Disk forensics techniques can be used to extract a variety of metadata (e.g. a file's author or its modification, access and creation times) and digital evidence (e.g. emails, images, videos and documents) from a file system. There are several commercial tools that perform this type of analysis, such as EnCase Forensics (Guidance Software 2013). VMware uses delta files to track changes to the disks of VMs. In this case, changes in unallocated parts of the hard drive and modification of existing files will appear in these delta files, even if the file system is undisturbed. However, approaches to detect rootkits based on the delta files are beyond the scope of this work.

Memory forensics techniques can be used to gather system state information from volatile memory. Process and network connectivity information can be extracted from a memory image (Okolica & Peterson 2010). The Virtual Address Descriptor (VAD) data structure maintained by the kernel to keep track of allocated memory ranges can be used to enumerate the files and objects to which each process has references (Vömel & Freiling 2011, Russinovich, Solomon & Ionescu 2009). Aljaedi et al. (Aljaedi, Lindskog, Zavarsky, Ruhl & Almari 2011) discuss the large variety of information types able to be extracted from memory. A range of tools have now been developed to help with the analysis of volatile memory images, such as the Volatility Framework (Volatile Systems 2013).

In addition to forensics tools, memory and disk driver software on the VM monitor is often similar or identical to the driver software on the guest. In this case, the drivers on the monitor can be used as a template for reconstruction the memory and disk structures. This is the approach used in VMWatcher (Jiang et al. 2010).

Derivative delivery (Pfoh et al. 2009) generates an OS-level view using knowledge of the virtual hardware architecture of VMs. An example of this approach is monitoring the contents of CPU control registers to infer information about the state of the VM. This approach is portable between different guest operating systems and is much less likely to be subverted by malicious modification of guest data structures. Unfortunately it is not portable across VM hardware architectures and can only extract limited information as prior semantic knowledge of the guest operating system cannot be used as a part of this approach.

2.2 Existing Implementations

View comparison based rootkit detection was first implemented by Strider GhostBuster (Wang et al. 2005), which compares the high-level view provided by the Win32 APIs to the lower-level view obtained by parsing data structures directly. These data structures are accessed by using a kernel driver, triggering a kernel memory dump or booting the monitored system from a WinPE boot CD. GhostBuster is able to detect hidden files, processes, kernel modules and registry entries. The file and registry view comparison aspects of this tool were subsequently released as RootkitRevealer, which used a kernel driver for its trusted view (Microsoft Research 2010).

Kernel Rootkit Trojan Detector (KeRTD) (Mahapatra & Selvakumar 2011) uses a number of techniques to detect and defend against rootkits. View comparison is used to compare process, driver and access control lists between kernel and user-mode to detect hidden objects.

A number of academic hypervisor-based IDS prototypes use VMI to provide the external view for their view comparison modules. Livewire (Garfinkel & Rosenblum 2003), XenKIMONO (Quynh & Takefuji 2007), VMwatcher (Jiang et al. 2007) and VMDetector (Wang et al. 2011) all read VM memory directly from the hypervisor to access kernel data structures. The contents of data structures listing kernel modules, processes and network connections are then compared to user-mode views of the corresponding objects to detect hiding.

Lycosid (Jones et al. 2008) uses a different approach to detect hidden processes. Process creation and deletion is inferred by observing related events, such as virtual address space creation and destruction, from the hypervisor. A hidden process causes the process count observed using this technique to differ from that reported by the guest operating system, allowing the hiding to be detected.

2.3 Our Implementation

We have developed a tool, named Linebacker, to perform view comparison and enable the use of standard disk and memory forensics on running VMs hosted on standard VMWare ESX servers. To our knowledge, none of the existing view comparison tools are designed to run on VMware enterprise infrastructure; VMWatcher is able to perform view comparison on VMWare Workstation though development of this capability required access to VMWare source code. This tool is able to compare internal (i.e. in-band delivery) and external (i.e. out-of-band delivery) views of memory, filesystem and registry objects for Windows VMs. Linebacker relies on VMware supported APIs to obtain the internal and external views required to perform view comparison. This approach eliminates the need to install an additional agent within each monitored VM, relying instead on installation of the standard set of VMware Tools within the Guest.

The view comparison techniques we employ aim to detect hiding of malware artefacts from the guest operating system. Examples include filtering the results of API calls before they are returned to the operating system or modifying kernel data structure using DKOM. Hide In Plain Sight (HIPS) techniques such as creating a file or process with a legitimate looking name or setting system or hidden attributes on files to conceal them from the user are not the target of our view comparison system, but may be detected by it. Such techniques do not prevent the guest operating system from accessing the concealed objects, and are better addressed through analysis of the visible file objects in the operating system than by view comparison.

Linebacker obtains an external view of memory objects by parsing Virtual Machine Suspended State (VMSS) memory images. VMSS files are generated by VMware hypervisors when VMs are suspended. These files store the metadata required to resume the execution of suspended VMs. This metadata necessarily includes an atomic snapshot of each VM's memory at the time is was suspended. The VMware vSphere Web Services API (VMware 2014*c*) is used to momentarily suspend each monitored VM, triggering the creation of a VMSS file. The list of processes running in the VM is then parsed from this memory image using the open-source Volatility Framework (Volatile Systems 2013). This list is then compared to the corresponding internal list, which is obtained via the VMware Tools guest addition using the VMware VIX API (VMware 2014*b*). Any processes that appear only in the external view are reported as hidden.

Similarly, Linebacker obtains an external view of disk objects by parsing disk snapshots. The VMware vSphere Web Services API is used to capture a snapshot of each monitored VM. These snapshots are then mounted using the VMware Virtual Disk Development Kit (VDDK) (VMware 2014*a*), allowing filesystem-level access to the contents of the VM's disks. This allows the files and directories on each disk to be enumerated and compared to the corresponding internal view. The internal view is obtained by using the VIX API to execute the `dir` command within the guest operation system. This command is run with flags to include any hidden files and to run recursively through all subdirectories. Any files that appear only in the external view are reported as hidden.

Linebacker also uses disk snapshots to reconstruct an external view of the non-volatile contents of the Windows registry (Russinovich, Solomon & Ionescu 2012). This is done by parsing the 'hive' files storing the registry using Hivex (Jones 2010). The corresponding internal view of the registry is obtained by using the VIX API to recursively enumerate all keys and values in the registry using the `RegEnumKeyEx` and `RegEnumValue` Windows API functions. These internal and external views can then be compared to detect hidden registry entries.

3 Detection

We conducted a set of tests in order to determine the efficacy of Linebacker's view comparison techniques at detecting the evasion behaviour of rootkits. This involved using Cuckoo Malware Sandbox in conjunction with our own scripts to execute Linebacker and Cuckoo on a corpus of malware executables. Each sample was run on a Windows XP x86 VM and a Windows 7 x86 VM hosted on the VMware ESXi 5.5 hypervisor.

While Cuckoo is not specifically designed to detect rootkits, it is designed for wider analysis of rootkits. It does not use the same techniques to extract information as rootkit detectors, as it sandboxes the execution of malware rather than using API hooking (i.e. it does not install a system level driver that manipulates or reads API calls at a kernel level), but it provides the same type of information as is used by rootkit detectors such as GMER (*GMER - Rootkit Detector and Remover* 2014). As an analysis tool rather than a detector, it will provide all the metadata it can about the malware it is sandboxing without filtering this information down to a set of single alerts for each rootkit detected. Our position is that the most informative way of analyzing view comparison as a rootkit detection technique is to compare the total sets of raw data obtained by Linebacker and Cuckoo when tested against a broad range of object hiding techniques, then examining both (a) what type of techniques are detected by each, and (b) how this raw data could be filtered to produce a rootkit detector. If a filtered output of Linebacker was compared to a end-user tool such as GMER, it would be harder to determine if any differences in performance were due to view comparison or differences in filtering techniques. Also, given the constant changes in malware, a relative comparison of the types of object hiding detected by Linebacker and Cuckoo is more informative than comparing the total number of rootkits detected by each.

3.1 Method

The method used to conduct the tests was:

1. reset the VM-under-test to a "base" state (i.e. has the Cuckoo agent running and no malware sample executing)
2. execute Linebacker's registry, disk and memory view comparison tools on a clean environment to record metadata about the "base" state of the VM
3. use Cuckoo to inject the malware sample and execute it in a sandboxed manner
4. use Cuckoo to record all metadata about the malware executable using the Cuckoo agent
5. execute Linebacker's registry, disk and memory view comparison tools on the infected VM to record its "dirty" state (i.e. potentially hidden files, processes and registry keys)
6. export all reports for later analysis
7. power off the VM.

This method was repeated for each of our malware samples, against Windows XP and Windows 7 SP1 VMs-under-test. The malware samples used for this testing consisted of the 13 rootkits listed in Table 1. We chose a range of 40 publicly-available rootkits that were representative of a range of object hiding techniques used by rootkits encountered on the internet. These techniques are summarised in Section 2.3. We then ran them on our test Windows XP system, and removed those that did not run, or ran but had no effect on any aspect of the system (as measured by Cuckoo). In the case of the FUTo(Silberman 2006) rootkit, the version that we found did not run on our Windows XP test system, but we were able to find a cut-down version of the rootkit that employs the same techniques and ran successfully on Windows XP. This was tested later.

The information provided by Cuckoo includes:

- native function and Windows API call traces
- copies of files created and deleted from the file system
- child process and process activity logs.

We use Linebacker's view comparison techniques to provide metadata about:

- processes for which an `EPROCESS` structure can be parsed directly from an external VMSS memory image, but which are omitted from the results of Windows API calls used to enumerate processes internally;
- files that are visible when directly parsing an external snapshot of a VM disk, but are not listed internally;
- registry keys that are visible when directly parsing registry hive files from disk snapshots, but are not visible via the Windows API calls used to enumerate registry contents internally.

To compare the results of both techniques, we filtered out clear false positives from each set of results. The baseline registry, process and file system chatter were removed from Linebacker by removing any result that appeared in both the clean and dirty runs of Linebacker for a particular rootkit. In addition,

the dirty results for every test included malware files uploaded to the sandbox by Cuckoo and Windows prefetch files created as a result of executing the malware. These files were also filtered from the results. To determine if the remaining artefacts were part of the rootkit or unrelated, we consulted available references on the behaviour of each rootkit. Where necessary, we repeated the test procedure and manually inspected the "dirty" state of the VMs through a VMware console. Through this, we could determine as precisely as possible how much of the information provided by Cuckoo or Linebacker was directly related to the detection of a rootkit.

3.2 Results

The detection testing results obtained using Linebacker are summarised in Tables 1 and 2. For Windows 7, we have also indicated whether, based on any evidence of activity in Cuckoo, the rootkit has installed successfully. The results are described below, and analysed in more detail in Section 3.3

Note: We refer to the memory view comparison results as "process" results, as we were only examining potentially hidden process information rather than all of the artefacts that could be extracted from memory.

Process Results: While Linebacker's process view comparison tool detected three true positive hidden processes, a large number of false positives were removed through the filtering and verification process. Both Linebacker and Cuckoo detected a hidden process for the `zeus.melt.exe` rootkit on both editions of Windows tested. The name of the process is actually a random string, which changed each time we executed the rootkit. This behaviour is consistent with published descriptions of this rootkit's behaviour. A hidden instance of `smss.exe` was also detected for two samples from the Zbot family, when run on Windows XP. Cuckoo did not detect this. On Windows 7, these two samples were not successfully installed.

As stated above, the FUTo rootkit was executed manually on both Windows XP and Windows 7. It was used to hide the `notepad.exe` process. On Windows XP the hidden process was detected by Linebacker. On Windows 7, the rootkit failed to run.

Disk Results: After filtering the results from Linebacker, and comparing the results with references on these rootkits, it was found that Linebacker had detected only two of the malware samples, `Trojan-Spy.Win32.ZBot.dol.exe` and QVOD, hiding files. This sample hid the directory `C:\Documents and Settings\Test\Application Data\wsnpoem` in Windows XP. This directory contained two files: `audio.dll` and `video.dll`. The `wsnpoem` directory is created by particular variants of the Zeus trojan. The `audio.dll` file within this directory is used to store stolen data to be sent back to the command and control server. The `video.dll` file stores the encrypted configuration for the trojan (ZeusTracker 2014). Linebacker also detected these files in Windows 7, in the directory `C:\Users\testadmin\-Roaming\wsnpoem`, though in this case Cuckoo also detected the files. Cuckoo observed the creation of three executable files by three variants of the Zeus/Zbot trojan: `oembios.exe`, `ntos.exe` and `sdra64.exe`, but did not detect the creation or presence of the files found by Linebacker on the Windows XP test machine. Linebacker did not detect the executable files found by Cuckoo as being hidden.

For the QVOD rootkit, a number of files were detected by Linebacker within the system restore directories. A manual verification revealed that these files were detected because the rootkit had set the "system" attribute for these files. This prevents them from being read by the `dir /h` command for anyone but the system administrator. While this is an example of Linebacker detecting rootkit activity, these files are only hidden from some users of the operating system, as opposed to the files hidden by `Trojan-Spy.Win32.ZBot.dol.exe`.

Linebacker detected files from other samples, on both Windows XP and Windows 7 test machines, but it was discovered during the verification process that these files were not associated with the malware under test. These files were either hidden from Linebacker's internal view as a result of filesystem permissions, hidden with the "system" attribute as described above, or created in the time interval between the internal and external views of disk being generated.

Registry Results: The registry view comparison did not detect any hidden keys or values.

3.3 Discussion

Process Results: The large number of false positives removed through the filtering and verification process can be attributed to the small delay between capturing the memory image providing the external process listing and the generation of the internal view via VMware Tools. In practice it is not possible to synchronise these two views perfectly as a VM must be paused to allow an atomic memory image to be captured. We expect the number of false positives to increase with process creation and termination activity on the monitored system.

For example, a process in the system may terminate during the time it takes to download the VMSS memory image from the guest. This process would appear in the external memory image, but would not be reported in the internal process listing. Our tool would then detect this process as potentially malicious, even though it is simply the result of benign process creation and exit activity.

Another cause of process false positives is `EPROCESS` structures for terminated processes remaining in memory. These structures remain in memory until they are overwritten or wiped, which may not happen for some time after a process terminates. Such structures will be parsed as active processes in the external view, resulting in a false positive.

To our knowledge the false positives would be difficult to filter outside of a controlled test environment, as filtering could also filter out process hiding that a rootkit may use to preserve itself. Our results show the concept of memory-based view comparison has the potential to work. However it is our recommendation that a realistic security solution would require cross checking against additional detection capabilities (such as live in-memory monitoring of process behaviour — something that is not possible using the free VMware APIs).

Disk Results: Linebacker's disk analysis detected object hiding from only one of the rootkits in the test set. The files created by this rootkit were not observed by Cuckoo. Though Cuckoo's recording of API callsg found more rootkits, and found more files associated with rootkits overall, this suggests that view comparison and tracking API calls may work as complementary approaches to rootkit detection.

As with the memory analysis, the disk view comparison produced a large number of false positives. An advantage of the disk view comparison technique is that false positives tend to appear in the same locations in the filesystem. False positives that occur

Table 1: Summary of hidden objects detected by Linebacker (Windows XP x86)

Rootkit	Installed?	Process	File	Registry
Backdoor.Win32.Ghost.binder.exe	Y	-	-	-
Backdoor.Win32.Ghost.20.exe	Y	-	-	-
Backdoor.Win32.Ghost.21.exe	Y	-	-	-
Backdoor.Win32.Ghost.23.exe	Y	-	-	-
QVOD	Y	-	Y	-
TDL4	Y	-	-	-
Trojan-Spy.Win32.Zbot.dol.exe	Y	-	Y	-
Trojan-Spy.Win32.Zbot.aaak.exe	Y	smss.exe	-	-
Wink.bg.exe	Y	-	-	-
zbot.exe	Y	smss.exe	-	-
ZeroAccess-z (dumped.dll)	Y	-	-	-
zeus.melt.exe	Y	meyw.exe	-	-
FUTo(manual)	Y	notepad.exe	-	-

Table 2: Summary of hidden objects detected by Linebacker (Windows 7 x86)

Rootkit	Installed?	Process	File	Registry
Backdoor.Win32.Ghost.binder.exe	Y	-	-	-
Backdoor.Win32.Ghost.20.exe	N	-	-	-
Backdoor.Win32.Ghost.21.exe	N	-	-	-
Backdoor.Win32.Ghost.23.exe	Y	-	-	-
QVOD	N	-	-	-
TDL4	Y	-	-	-
Trojan-Spy.Win32.Zbot.dol.exe	Y	-	Y	-
Trojan-Spy.Win32.Zbot.aaak.exe	N	-	-	-
Wink.bg.exe	Y	-	-	-
zbot.exe	N	-	-	-
ZeroAccess-z (dumped.dll)	Y	-	-	-
zeus.melt.exe	Y	leby.exe	-	-
FUTo(manual)	N	-	-	-

as a result of file permissions can be filtered out using a fixed list of directories. File hiding that occurs as a result of rootkits, and appears in other directories, is unusual and flagging such activity should produce few false positives. A drawback of this approach to reducing the number of false positives is that an attacker could design malware that hides in parts of the system that have the highest level of security. Unless Linebacker is run with this level of access, files hidden in these areas will be assumed to be hidden via normal system permissions rather than hidden by an attacker. Running Linebacker with administrator credentials on all the systems it monitors could be seen as creating an additional method for these systems to be compromised.

A key issue with file-based view comparison is that it only detects rootkits that exhibit a certain class of file-hiding behaviour. Such behaviour is far from ubiquitous. Unsophisticated rootkits may not make any attempt to avoid detection, thus view comparison will not detect them. Rootkits that employ HIPS strategies will not be detected for similar reasons. Some modern rootkits, such as TDL-4, use alternative evasion techniques that avoid creating file system objects entirely. Later versions of the TDL rootkit store the exploit in slack space at the end of a drive, then overwrite the MBR to load the exploit before the rest of the operating system (Lau 2013). This technique circumvents the Master File Table (MFT), so the view comparison approach will not detect these files — they will not appear in internal or external analyses of the file table. However, forensic analysis of the external view of the disk could detect changes in slack space.

Despite these issues, view comparison does detect some rootkits and provides information about where they hide on the system, making it a valuable addition to an arsenal of complementary detection tools. However, if a new system for monitoring VMs were to be developed, an approach based on offline full analysis of disk changes would be able to detect a wider range of threats than view comparison alone.

Registry Results: Our registry view comparison detected only a small number of entries. These appeared in both the "clean" and "dirty" scans, so they are not the result of object hiding by rootkits. Cuckoo observed that registry entries were created by some of the rootkits in our sample. We believe that these entries were not hidden as there is little benefit to cloaking registry entries from the guest operating system. Cloaking these entries would prevent them from having their intended effect, such as automatically re-launching the rootkit following a reboot.

Overall: The Linebacker tools detected a small proportion of the rootkits we tested. These rootkits use a range of techniques to prevent their detection, from HIPS through DKOM to hiding in slack space. The view comparison technique is only designed to detect a subset of these techniques. To get protection against a broad band of rootkits, Linebacker would need to be combined with other detection techniques such as traditional antivirus techniques and MBR

protection techniques. The Linebacker tools tend to generate a large number of false positives that must be filtered out in some way. The registry and hard drive in particular have large amounts of ongoing change and deletion even on VMs that are only running the malware under test. The inability to perfectly synchronise the external and internal views makes it difficult to determine (without manual verification) whether or not a process detected by the memory analysis tool is a false positive or not. File permissions and UAC in Windows 7 can also cause large numbers of false positives (i.e. preventing Linebacker's internal view from accessing files which are not actually hidden in a malicious sense). This is highlighted by the fact that our testing against known malware samples generated thousands of lines of marked-as-hidden output that needed to be filtered out before meaningful analysis could take place. This could potentially be an even greater problem for scaling view comparison up from single machines to enterprise systems, though in enterprise systems there is the potential to compare output across a set of identical systems to identify the anomalous behaviour caused by rootkits (Bianchi, Shoshitaishvili, Kruegel & Vigna 2012).

4 Performance Case Study

We conducted a series of tests to determine the performance of the Linebacker view comparison tools on a small scale set of production VMware ESXi infrastructure. These tests involved recording execution time metrics for the memory and registry components of Linebacker. Due to limitations of the older version of VMware ESXi used on our infrastructure, we were not able to run the memory tool's view comparison techniques, nor could we execute the disk analysis tool. This is because these tools rely on later versions of the VMware Web Services SDK.

Our tests monitored a set of 9 VMs running the Windows 7 operating system, which were hosted on a cluster of VMware ESXi 4.1 servers running on IBM HS22V blades. The VMs were used for general office work by volunteer members of our research group for one month.

Memory: The time taken to prepare the analysis of memory from a series of VMs varied erratically throughout the analysis based on current network and server loads. A single preparation time includes the time taken to suspend and resume a VM-under-analysis and then download its VMSS file for analysis. The total time for all VMs to be prepared in a memory analysis batch job and the time taken to analyse the memory snapshots are given in Table 3. The high standard deviation for the total batch time highlights how erratic the performance of VMware API calls can be. We suggest that this behaviour is caused by surges in network activity slowing the VMSS download or heavy server disk activity slowing the time taken to suspend the VM and then write its VMSS file to disk.

The relatively low times for processing snapshots show that the offline memory analysis is not the performance bottleneck. This is largely expected as Linebacker uses the well tested and robust Volatility framework to extract memory artefacts from the VMSS files by efficiently parsing in memory Windows kernel structures. This suggests that if it were possible to analyse the memory images in place, large performance gains could be made.

There are noticeable performance impacts that arise as a result of capturing an up-to-date memory snapshot for the purpose of forensically examining it via the Linebacker memory tool. These arise as a result of ESXi hosts only creating a memory dump when a VM is suspended. As the entire memory is imaged in this process, the time taken to suspend then resume a VM is related to the size of the memory provided to that VM. The time taken is also related to the current disk and CPU load on the ESXi host. We received several reports from users of noticeable hitching or pausing as a result of the memory analysis tool.

It is worth noting that performance impacts are worse in ESXi versions 5.5 and later, as they immediately delete VMSS files upon resuming from a suspended state. In versions 5.1 and earlier the VMSS file lingered after the resuming of the VM and could downloaded in the background for later analysis. In more recent versions, the VMSS file can only be captured to disk if the VM is paused for the duration of the capture time. This means that the execution of monitored VMs must be halted while the memory snapshot is created and downloaded from the hypervisor.

Our suggestion would be that given the performance impacts of capturing the up-to-date memory snapshot and the time taken to capture it, this analysis would only be run at off-peak times.

Disk and Registry: The disk and registry components of Linebacker rely on a single shared disk snapshot for their external view. Capturing snapshots had a negligible impact on users of the VMs in our trial. In some cases a small pause in execution lasting less than a second was observed by users during snapshot creation, although most users noticed no impact on the execution of their VMs at all.

The analysis time for the registry component of Linebacker includes generation of an internal registry view, capturing a disk snapshot and comparing the external registry view stored in the snapshot with the internal view. We recorded the total batch time to perform this analysis for all VMs in our case study, as well as the time taken to perform each step of this analysis on individual VMs. These times are summarised in Table 3.

We consider the time taken to analyse all VMs to be reasonable given the number of VMs being monitored. We believe the time taken to perform individual analysis steps shows that this approach can feasibly be implemented in production environments. While monitoring larger systems of hundreds or thousands of VMs was outside the scope of our case study, we expect that a scalable implementation could be achieved by performing the monitoring in a distributed manner across the system.

It is important to remember that the execution of monitored VMs continues throughout the disk and registry monitoring process, with the possible exception of a small pause at the start of the snapshot creation process.

5 Conclusions

Our efficacy results show that virtualisation assisted view comparison detects a limited subset of modern rootkits. We attribute this to malware authors either hiding objects outside the file system, or taking a HIPS approach and relying on the presence of other filesystem activity to hide the activities of their malware. Linebacker detected six of thirteen rootkits on Windows XP, and two of seven rootkits on Windows 7. However, five of the rootkits tested used no object hiding. Two rootkits hid rootkit objects by altering existing files, registry entries or processes rather than creating new ones. One rootkit hid objects outside

Table 3: Performance case study results (in seconds)

Operation	Mean Time	Median Time	Std. Dev.
Memory batch preparation (all VMs)	1202	715	1018
Analyse memory snapshot (per VM)	21.48	20.57	3.14
Batch registry analysis (all VMs)	787	837	274
Internal registry view (per VM)	11.3	11.0	3.9
Capture disk snapshot (per VM)	14.9	16.0	4.3
Registry view comparison (per VM)	56.5	55.0	11.5

the filesystem. Each rootkit that was able to hide new files or processes without going outside the filesystem to do so was detected by Linebacker.

The registry analysis component detected no rootkit evasion behaviour. We suggest that this is likely caused by the fact that authors can effectively maintain a persistent key in the registry without resorting to installing filter drivers or applying other evasion tactics. There is a large amount of existing noise in the registry due to general Windows OS activity. Hiding in this background "white noise" is relatively easy. Furthermore, employing evasion techniques such as API manipulation within the registry is risky due to the high volume of activity in the registry. A filter driver applied here by a rootkit is likely to noticeably degrade system performance, which may arouse the suspicion of users or administrators.

The disk analysis detected little rootkit evasion behaviour. With several rootkits not using object hiding, and one rootkit hiding in slack space, there were few cases where rootkits used hiding techniques detectable by Linebacker.

The memory analysis component did successfully identify rootkit evasion behaviours, however we found it difficult to discern true positives from false positives without manual analysis of the target VM. This was primarily due to processes starting and stopping between internal and external scans, and `EPROCESS` structures remaining resident in memory for an indeterminate time. We attempted to rectify this by checking the process end time field in this structure, however we have found that the field is often left empty.

There are fundamental issues with the use of view comparison to detect rootkits. The difficulties we experienced in synchronising the internal and external views of disk and memory cannot be resolved without modifying the hypervisor to support more direct access to the state of executing VMs. A large number of false positives are caused by this sync issue, in addition to those caused by file access permissions. Finally, a large number of rootkits, as shown by our sample set, either employ a HIPS strategy or use alternative hiding techniques that our tool is not designed to detect.

Our performance results highlight that disk and registry view comparison and VMI techniques can be implemented on VMware vSphere infrastructure with minimal impact. We also identified that memory-based process view comparison and VMI causes noticeable hitching and performance issues on monitored VMs. These issues arise from the need to suspend VMs in order to create up-to-date VMSS files. Unfortunately, creating these VMSS files cannot be avoided as they contain the atomic raw memory snapshot required for perform view comparison analysis on a VM.

If it was to be used in rootkit detection, view comparison would be best used as part of a complementary set of approaches. In spite of the issues with view comparison, we feel that VMI approaches as a whole provide a valuable source of "trusted" external view computer security audit data. That is, VMI and the VMware APIs provide an accurate view of the processes, network connections, sockets, files and registry contents on VMs. VMI may be a good basis for detecting object hiding techniques that alter or work outside the filesystem itself, in addition to those detected by view comparison. Memory-based VMI causes performance impacts in vSphere deployments and as a result care must be taken in determining when the analysis will take place. The disk and registry VMI tools can be applied in vSphere infrastructure with minimal impact.

Our plan in future is to use Linebacker as a basis for analysis of multiple machines across a network. In this case, rather than performing analysis within Linebacker, it will act as a data source for existing analytic tools. By comparing multiple similar machines on the same network, we hope to identify anomalous behaviour in single VMs and track how this behaviour moves through a network of VMs.

References

Adventium Labs (2014), 'Virtual cyber defender introspection appliance (vcd-ia)'.
URL: *http://www.adventiumlabs.com/our-work/products-services/virtual-cyber-defender-introspection-appliance-vcd-ia*

Aljaedi, A., Lindskog, D., Zavarsky, P., Ruhl, R. & Almari, F. (2011), Comparative analysis of volatile memory forensics: Live response vs. memory imaging, *in* 'Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom)', pp. 1253 –1258.

Bianchi, A., Shoshitaishvili, Y., Kruegel, C. & Vigna, G. (2012), Blacksheep: detecting compromised hosts in homogeneous crowds, *in* 'Proceedings of the 2012 ACM conference on Computer and communications security', CCS '12, ACM, New York, NY, USA, pp. 341–352.
URL: *http://doi.acm.org/10.1145/2382196.2382234*

Bitdefender (2014), 'Bitdefender security for virtualized environments'.
URL: *http://enterprise.bitdefender.com/solutions/gravityzone/virtualization-security.html*

Cuckoo Sandbox Developers (2014), 'Cuckoo Sandbox'.
URL: *http://www.cuckoosandbox.org*

Garfinkel, T. & Rosenblum, M. (2003), A virtual machine introspection based architecture for intrusion detection, *in* 'Proc. Network and Distributed Systems Security Symposium', pp. 191–206.

GMER - Rootkit Detector and Remover (2014). Viewed 16/05/2013.
URL: *http://www.gmer.net*

Guidance Software (2013), 'EnCase Forensic: Computer forensic analysis software'.
URL: *https://www.encase.com*

Hay, B. & Nance, K. (2008), 'Forensics examination of volatile system data using virtual introspection', *SIGOPS Oper. Syst. Rev.* **42**, 74–82.

Jiang, X., Wang, X. & Xu, D. (2007), 'Stealthy malware detection and monitoring through VMM-based 'out-of-the-box' semantic view reconstruction', *ACM Trans. Inf. Syst. Secur.* **13**, 12:1–12:28.

Jiang, X., Wang, X. & Xu, D. (2010), 'Stealthy malware detection and monitoring through vmm-based “out-of-the-box” semantic view reconstruction', *ACM Trans. Inf. Syst. Secur.* **13**(2), 12:1–12:28.
URL: *http://doi.acm.org/10.1145/1698750.1698752*

Jones, R. (2010), 'Hivex - Windows registry "hive" extraction library'. Viewed 26/07/2013.
URL: *http://libguestfs.org/hivex.3.html*

Jones, S. T., Arpaci-Dusseau, A. C. & Arpaci-Dusseau, R. H. (2008), VMM-based hidden process detection and identification using Lycosid, *in* 'Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on virtual execution environments', VEE '08, ACM, pp. 91–100.

Kapoor, A. & Mathur, R. (2011), Predicting the future of stealth attacks, *in* 'the 21st Virus Bulletin International Conference'.

King, S. T. & Chen, P. M. (2005), 'Backtracking intrusions', *ACM Trans. Comput. Syst.* **23**, 51–76.

Krishnan, S., Snow, K. Z. & Monrose, F. (2010), Trail of bytes: efficient support for forensic analysis, *in* 'Proceedings of the 17th ACM Conference on Computer and Communications Security', CCS '10, ACM, pp. 50–60.
URL: *http://doi.acm.org/10.1145/1866307.1866314*

Lau, H. (2013), 'Backdoor.Tidserv - technical details'.
URL: *http://www.symantec.com/security_response/writeup.jsp?docid=2008-091809-0911-99&tabid=2*

Litty, L., Lagar-Cavilla, H. A. & Lie, D. (2008), Hypervisor support for identifying covertly executing binaries, *in* 'Proceedings of the 17th Conference on Security Symposium', USENIX Association, pp. 243–258.

Mahapatra, C. & Selvakumar, S. (2011), 'An online cross view difference and behavior based kernel rootkit detector', *SIGSOFT Softw. Eng. Notes* **36**(4), 1–9.

Microsoft Research (2010), 'Strider GhostBuster rootkit detection'.
URL: *http://research.microsoft.com/en-us/um/redmond/projects/strider/rootkit/*

Nance, K., Hay, B. & Bishop, M. (2009), Investigating the implications of virtual machine introspection for digital forensics, *in* 'Availability, Reliability and Security, 2009. ARES '09. International Conference on', pp. 1024 –1029.

Okolica, J. & Peterson, G. L. (2010), 'Windows operating systems agnostic memory analysis', *Digital Investigation* **7, Supplement**(0), S48 – S56. The Proceedings of the Tenth Annual DFRWS Conference.

Pfoh, J., Schneider, C. & Eckert, C. (2009), A formal model for virtual machine introspection, *in* 'Proceedings of the 1st ACM Workshop on Virtual Machine Security', VMSec '09, ACM, pp. 1–10.

Quynh, N. A. & Takefuji, Y. (2007), Towards a tamper-resistant kernel rootkit detector, *in* 'Proceedings of the 2007 ACM Symposium on Applied Computing', SAC '07, ACM, New York, NY, USA, pp. 276–283.

Russinovich, M. E., Solomon, D. A. & Ionescu, A. (2009), *Windows Internals*, fifth edn, Microsoft Press.

Russinovich, M., Solomon, D. & Ionescu, A. (2012), *Microsoft Windows Internals: Part 1*, 6th edn, Microsoft Press.

Silberman, P. (2006), 'FUTo'.
URL: *http://uninformed.org/index.cgi?v=3&a=7*

Sophos (2014), 'Sophos antivirus for vShield'.
URL: *http://www.sophos.com/en-us/products/server-security.aspx*

Trend Micro (2014), 'Deep security'.
URL: *http://www.trendmicro.com.au/au/enterprise/cloud-solutions/deep-security/*

VMware (2014*a*), 'VDDK documentation'.
URL: *http://www.vmware.com/support/ developer/vddk/*

VMware (2014*b*), 'VIX API documentation'.
URL: *http://www.vmware.com/support/developer/vix-api/*

VMware (2014*c*), 'VMware vSphere Web Services SDK documentation'.
URL: *http://www.vmware.com/support/developer/vc-sdk/*

VMware (2014*d*), 'vShield Endpoint'.
URL: *http://www.vmware.com/au/products/vsphere/features-endpoint*

Volatile Systems (2013), 'The Volatility framework: volatile memory artifact extraction utility framework'.
URL: *https://www.volatilesystems.com/default/volatility*

Vömel, S. & Freiling, F. C. (2011), 'A survey of main memory acquisition and analysis techniques for the windows operating system', *Digital Investigation* **8**(1), 3–22.

Wang, Y., Hu, C. & Li, B. (2011), VMDetector: A VMM-based platform to detect hidden processes by multi-view comparison, *in* 'High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on', pp. 307 –312.

Wang, Y.-M., Beck, D., Vo, B., Roussev, R. & Verbowski, C. (2005), Detecting stealth software with Strider Ghostbuster, *in* 'Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on', IEEE, pp. 368–377.

Wang, Y.-M., Vo, B., Roussev, R., Verbowski, C. & Johnson, A. (2004), Strider ghostbuster: Why it's a bad idea for stealth software to hide files, Technical report, Technical Report MSR-TR-2004-71, Microsoft Research.

ZeusTracker (2014), 'Zeus tracker - faq'.
URL: *https://zeustracker.abuse.ch/faq.php*