2010 39th International Conference on Parallel Processing Workshops

A Case for Secure Virtual Append-only Storage for Virtual Machines

Zhao Lin, Kartik Gopalan, Ping Yang
Department of Computer Science
State University of New York at Binghamton,
Binghamton, NY, 13904, USA
Contact: {kartik,pyang}@binghamton.edu

***Abstract*—Traditional operating systems and applications use logs extensively to monitor system activity and perform intrusion detection. Consequently, logs have also become prime targets for intruders. When a malware or intruder obtains root privileges in a system, one of its first actions is to hide its footprint by deleting or modifying system logs, especially the log entry recording the intrusion activity (such as unauthorized root login). A key weakness of most current logging mechanisms is that logs are stored on a storage device over which the system being logged has complete control, including the ability to delete/modify the logs arbitrarily. Once the root privileges of such a system are compromised, so are the logs. Virtualization offers a unique opportunity to eliminate this point of weakness.**

In this paper, we propose a new virtual storage abstraction for virtual machines (VMs) called *Virtual Append-only Storage* (VAS) that secures and preserves all system and/or application logs in a VM and can prevent an intruder from deleting/modifying past logs even after the root privileges of a VM are compromised. Our VAS-based logging complements existing intrusion detection techniques which mainly monitor the in-memory execution state and data, but do not protect the storage device on which logs are stored. Since logs can become voluminous over time, VAS also provides administrators the ability to secure either system-wide or application-specific logs, rather than blindly logging all system activity.

I. Introduction

Cloud computing is being rapidly adopted in enterprise environments to improve IT efficiencies and increase operational and management flexibility. Cloud computing enables users to take advantage of remote computing resources to execute a large variety of demanding applications. Virtualization technology is being widely used in cloud computing environments to reduce space requirements, hardware complexity, and operating costs.

Implicit in the cloud model is a trust that users place in the cloud infrastructure to keep their data and code secure from unauthorized and malicious access. In this respect, virtualization also enhances the security of cloud applications by providing functional and performance isolation between independent virtual machines (VMs). In addition, virtualization allows administrators to deploy new security mechanisms that were previously not possible in non-virtualized settings. For instance, malware analysis and intrusion detection in VMs can be performed transparently [1], [2], [3], [4], [5], [6], [7] either from an external VM or from the hypervisor itself.

While securing cloud computing applications in virtualized settings has many facets, this paper addresses one piece of the puzzle, namely, securing the logs that record system activity. Traditional operating systems and applications use logs extensively to monitor system activity and perform intrusion detection. Consequently, logs have also become prime targets for intruders. When a malware or intruder obtains root privileges in a system, one of its first actions is to delete or modify system logs, especially the log entries that record the intrusion activity (such as an unauthorized root login).

A key weakness of most current logging mechanisms is that logs are stored on a storage device over which the system being logged has complete control, including the ability to delete/modify the logs arbitrarily. Once the root privileges of such a system are compromised, so are the logs. Virtualization offers a unique opportunity to eliminate this point of weakness.

In this paper, we propose a virtualization-based secure logging mechanism that preserves and secures system-wide and application-specific logs from being tampered by an intruder. Our technique relies on a new virtual storage abstraction for VMs, called the *Virtual Append-only Storage* (VAS), that can prevent an intruder from deleting/modifying past logs even after the root privileges of a VM are compromised. The key idea is to transparently redirect all the logs generated by the VM's applications and operating system to a special VAS device. The VM is given permissions to only append new logs to the end of the virtual storage, or read existing logs, but is not allowed to edit or delete any logs. VAS-based logging is completely transparent to applications and requires no changes to the core operating system executing in the VM. VAS will complement existing intrusion detection techniques which mainly monitor the in-memory execution state and data, but do not protect the storage device

1530-2016/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPPW.2010.15

on which logs are stored. Since logs can become voluminous over time, VAS also provides administrators the ability to secure either system-wide or application-specific logs, rather than blindly logging all system activity.

The rest of the paper is organized as follows. Section II details the threat model and assumptions behind our work. Section III describes the detailed architecture of VAS. Performance results are given in Section IV. Section V describes the future challenges and research directions in implementing a comprehensive VAS-based secure logging and storage mechanisms. Section VI compares VAS against related work and the concluding remarks appear in Section VII.

II. Threat Model and Assumptions

We assume the following adversary model for intruders. Once the VM is compromised, the intruder acquires the highest privilege level within the VM and has complete control over the VM's filesystem, operating system (kernel), and applications. The intruder is capable of surreptitiously executing malicious code in the operating system of the VM being monitored (such as by inserting kernel modules) and thus hiding its activities. Once the intruder gains complete control of VM's execution environment, it can potentially bypass any logging mechanism, including logs to VAS. The minimal design goals of our VAS mechanism are to (1) securely record any system activity of interest up to (and including) the point at which the intruder is able to compromise or bypass VM's logging mechanism, and (2) prevent the intruder from tampering with or destroying logs that have been already recorded in the VAS device. We also assume that a virtual machine monitor (VMM or hypervisor) and a controller VM (such as Domain 0 in Xen) together provide a trusted computing base and isolate the activities of one VM from another. VMs themselves are not part of the trusted computing base. This is a standard assumption in virtualization community (e.g. [3], [6], [8]). Presence of hardware-based security support (such as Intel Txt [9]), although not required for VAS, can be leveraged to continue secure logging even after the VM's root privilege is compromised. We also assume that the procedure for operating system boot-up within a VM is secure, such as through the use of TPM [10] technology, and that network connectivity is not enabled in the VM until after VAS has been configured.

III. Architecture of VAS-based Secure Logging Mechanism

One of the objectives of the proposed VAS system is to maintain transparency for applications as well as operating system. In particular, we do not want any modifications (code changes, recompilation, or relinking) in applications that use VAS for logging. Similarly, we do not wish to modify the core operating system kernel image in the VM. Consequently, VAS is implemented as a set of two kernel modules shown in Figure 1.

A. Transparency Through Pseudo Filesystem

Within the VM being logged, VAS appears as a **pseudo filesystem** – in other words, a special filesystem whose contents do not reside on the VM's virtual disk. Instead, all I/O operations on this pseudo filesystem are intercepted and re-directed to Domain 0, which is a privileged VM trusted by the hypervisor for system management activities. The proposed pseudo filesystem is similar in some sense to the /proc filesystem used in Linux for system management. For example, the VAS pseudo filesystem in Linux could be mounted under the /var/log directory – the standard location for most applications and the operating system to store their logs. Each file in the /var/log directory corresponds to a log file maintained by one or more applications. For example, /var/log/syslog is a log file maintained for system-wide logs. Similarly, /var/log/httpd.log is a log file maintained by Apache webserver; /var/log/mysql.log is maintained by the mysql database server; and so on.

The set of operations permitted on the the pseudo filesystem are restricted for all applications in the VM (including those with root privileges). In particular, (a) the *delete* operation is prohibited, (b) the *append* operation (writing to the end of file) is permitted, but writes to any offset other than the end of the file is prohibited, and (c) *renaming* a file to a location outside the pseudo filesystem is prohibited. All other operations are permitted as they would be in a normal filesystem. This includes creating a directory, creating a file, changing file permissions, and reading from a file. Note that creation of hard links across VAS and external mount points is prohibited to prevent applications from bypassing the above restrictions. For instance, one cannot create a hard link from /tmp to /var/log if the VAS device is mounted under /var/log.

B. VAS Split Driver

In order to secure the log entries as soon as they are generated, any operation on the files in the pseudo filesystem needs to be conveyed out of the VM to Domain 0 with minimal delay. This mechanism for conveying the log contents to Domain 0 is designed as a *VAS split driver*, which consists of two parts – the *front-end driver* and the *back-end driver*. The VAS front-end driver resides in the operating system of the VM and has minimal functionality. It simply delivers I/O requests on the pseudo filesystem from

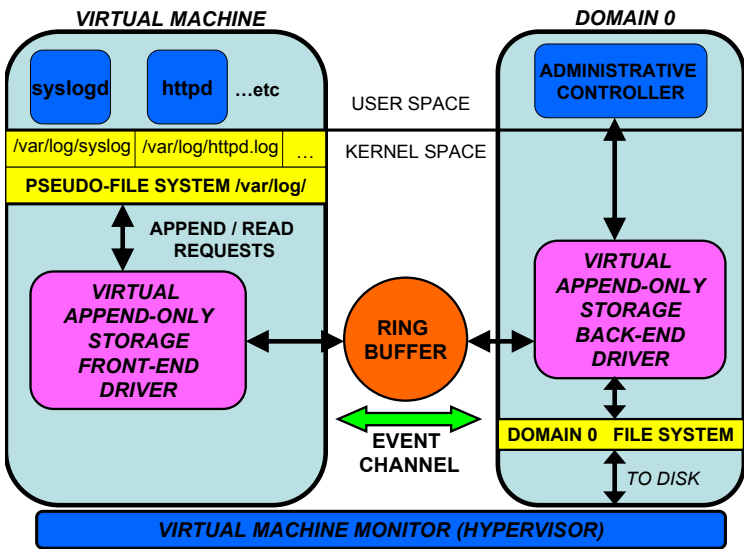

Figure 1. VAS-based Secure Logging Architecture.

the VM's applications to the VAS back-end in Domain 0 and the I/O responses from the VAS back-end to the applications. The VAS-backend in Domain 0 stores the contents of the VM's VAS pseudo filesystem in the local disk managed by Domain 0. It interacts with the local disk through Domain 0's filesystem interface. Before committing and I/O activity to the disk, VAS back-end applies the restrictions on *delete*, *write*, and *renaming* operations mentioned above and returns an I/O error to the VAS front-end if such operations are attempted.

The VAS front-end and back-end communicate I/O requests and responses with each other through a *shared buffer*, which is a lockless shared memory producer-consumer buffer of unit size. In addition, an *event channel* between the front-end and back-end is used to notify each other of the presence of data in the shared buffer. Note that, in order to deny the intruder a chance to erase the logs that have not yet been committed, all log entries spend minimal time within the VM and are shuffled out quickly to Domain 0. In particular, there is no filesystem cache for the pseudo filesystem in the VM which could delay the log entries from being committed immediately. Consequently, even an attacker who has compromised the VM's operating system will be denied an opportunity to erase any log entries sitting in the cache. The shared buffer itself is of unit size and the data in this buffer is committed to storage as soon as the VAS back-end receives an event notification from the front-end. The most harm an attacker could do is to erase just one log entry before Domain 0 has an opportunity to pick it up from the shared buffer – an unlikely possibility given the highly precise timing window required (in the order of less than 50 μs).

C. Implementation of VAS Split Driver Back-end

The back-end of the VAS split driver attaches to shared buffer with the front end. It then starts a kernel thread that waits for incoming requests from the front-end. All incoming I/O operations from the VM's VAS front-end are redirected to a VM-specific target directory in Domain0's filesystem. The back-end checks whether the incoming I/O operations are allowed before committing the requests to the target directory. Permitted operations include open, close, create, read, write to the end of the file, createdir, readdir. Prohibited operations include write to any offset other than end of file, delete, and link/unlink. To receive notifications regarding an incoming I/O operation, the back-end split driver binds an event handler with an inter-domain event channel. When the front-end signals an event over the channel, the front-end event handler simply wakes up the kernel thread.

D. Implementation of VAS Split Driver Front-end

During the secure boot up phase, the front-end of VAS split driver creates a page to hold the ring buffer shared with the back-end. The front-end then binds an event handler with the event channel. The purpose of the event handler is to wake up any process waiting for a reply to its I/O request once the request is committed (or denied) by the back-end. Upon initialization, the front-end retrieves the listing

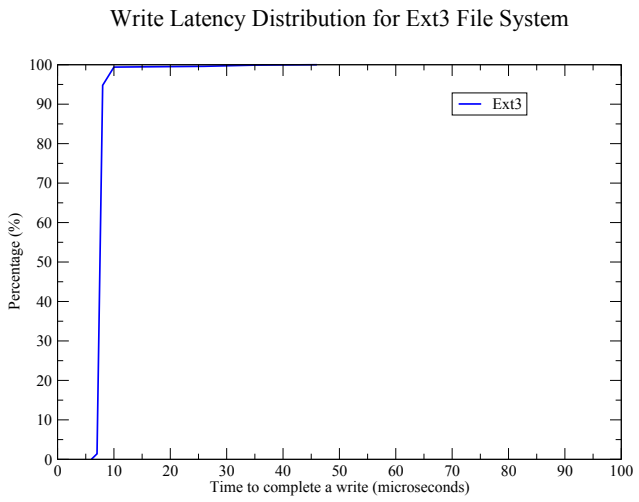

Figure 2. Write latency distribution for Ext3.

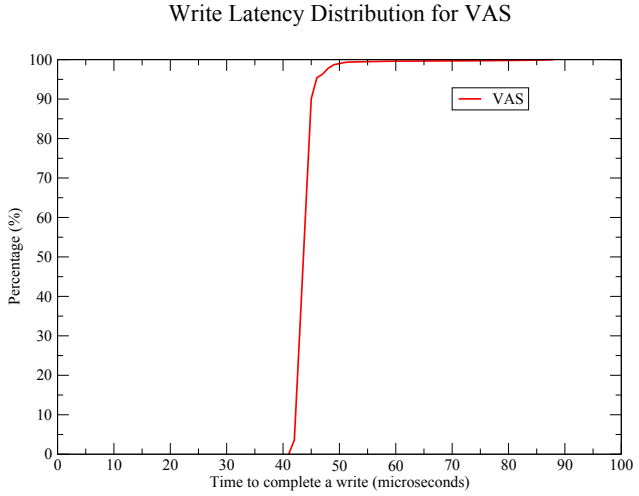

Figure 3. Write latency distribution for VAS device. Higher write latency results from the absence of a write buffer in the VM and the cross-domain communication to validate and commit write operations at Domain 0.

of existing directories and files in the VAS device from the back-end in Domain 0. The front-end then initializes and populates the pseudo filesystem with the list of directories and files retrieved in step 3. Whenever an application generates an I/O request (both legitimate and invalid) on the pseudo filesystem, the request passed on to the VAS back-end in Domain 0 through the ring buffer. The I/O operation blocks until Domain 0 replies back after either committing a legitimate request or by denying an invalid request.

IV. Performance Results

We evaluated the overhead of write I/O operations with our VAS device compared to a standard Ext3 filesystem in Linux. All experiments were conducted on a machine with Intel Dual Quad-core 2.3GHz CPU and 16GB of memory running Xen 3.3 and Linux 2.6.18. We focus primarily on writes since read operations are expected to be less frequent on event logs compared to write operations. Figure 2 shows the cumulative distribution function for the time it takes to complete a single write to a file in the Ext3 filesystem. The figure shows that 95% of write operations are completed within 8 microseconds for EXT3 filesystem. This is because the write system call returns immediately after the contents are written to the page buffers in the VM's Linux kernel. The write operations are committed to the disk asynchronously at a later time by the OS. Figure 3 shows the cumulative distribution function for the time it takes to complete a single write to a file in the pseudo filesystem on a VAS device. The figure shows that 95% of write operations are completed within 45 microseconds. The higher write latency compared to Ext3 results from the absence of a write buffer in the VAS front-end and the need to perform cross-domain communication across the ring-buffers (and the corresponding domain switching overhead) to validate and commit write operations at Domain 0.

To demonstrate the operation of VAS with real applications, we also set up an Apache web server in the guest VM and mounted the VAS-backed pseudo filesystem under /var/log directory in the VM. The Apache web server appends a message to /var/log/apache2/access.log for every client web page request. In a remote machine we run the benchmark `ab` to issue 10000 web page requests with concurrency level of 1. With the VAS device, the average time to complete each web access request is 0.723 ms. In comparison, when the Apache logs are placed on Ext3 filesystem, the average request completion time was found to be 0.648 ms. Thus Apache webserver is able to use more secure logs over a VAS device for an average request completion overhead of just under 11.6%.

V. Future Challenges and Directions

Our preliminary investigations of the VAS-based secure logging mechanism for VMs have pointed us to new research opportunities and challenges. We describe the specific research directions below.

A. Securing Log after Intrusion

Although the above VAS architecture can prevent an attacker in a VM from tampering or deleting any system logs after an intrusion, it does not prevent the attacker from bypassing the logging mechanism after the intrusion is successful. For instance, an attacker could replace the pseudo filesystem with a regular filesystem on the virtual

disk once the intruder gains root privileges. We are currently investigating techniques to protect the VAS logging mechanism from being bypassed after an intrusion. One possible approach is to use the hypervisor's *shadow paging* mechanism to mark all kernel pages that store VAS-related code, data, and entry points in the VM as read-only. Thus, any attempts by the intruder to modify or bypass the VAS front-end in the VM's operating system will generate a software exception in the hypervisor, which can then notify the system administrator.

Similarly, there is a short window of vulnerability between the time that front-end places an I/O request in the shared ring-buffer and the time the back-end picks up the I/O request. During this window, an intruder who has just compromised the VM could attempt to erase the log records in the ring-buffer that might potentially record the intrusion event. To prevent such malicious deletion, we are investigating the use of *shadow memory* to trap, and potentially deny, retroactive write attempts to the shared ring buffer. In effect, shadow memory enforces append-only semantics on the shared ring buffer.

B. Untamperable Virtual Storage

The basic concept behind VAS-based secure logging can be generalized to an *untamperable virtual storage* (UVS) system. The key idea is to develop a log-based storage interface for VMs in which all I/O operations are treated as log entries to an append-only virtual storage device. The key advantage of such a storage is that, post-intrusion, an attacker cannot permanently damage or destroy any prior data on the virtual disk because each subsequent operation is appended as a log entry to the virtual storage device. The main reason that such a protection is possible at all is that virtual machines do not have direct access to the physical storage devices. Rather, their I/O operations are redirected via their virtual storage interface to a special I/O domain (such as Domain 0 in Xen), which then commits the I/O operations to the disk. We can exploit this additional level of indirection to treat any new I/O operations as logs and to preserve past history of storage I/O operations. Even after an intrusion in the VM, any malicious I/O operations by an attacker will be committed as logs and can be easily rolled back to a safe state once the intrusion is detected. Since data is never actually deleted or overwritten, a key challenge is to minimize the amount of storage required to maintain I/O operations as logs.

C. Administrative Control Interface

It is important for the system administrator to monitor the state of the VAS for hundreds of VMs in the cluster and to configure their key parameters with little effort. We are developing two types of interfaces to simplify the task of the system administrator. First is a scripting API that can be used to automatically configure VAS parameters, boot-time setup, shutdown behavior, and intrusion monitoring. Second is a GUI front-end that can enable the administrator to monitor VAS status and deliver real-time updates.

D. Overhead Reduction

Although VAS-based secure logging protects system logs from being tampered with in the event of an intrusion, it achieves this functionality at the cost of an additional level of indirection via Domain 0. While this indirection overhead may be common for all virtual storage systems, it could be potentially more apparent for VAS since its pseudo filesystem in the VM does not carry a filesystem cache. Thus every I/O operation can potentially suffer from additional latency while being committed to the VAS back-end. We are investigating techniques to reduce this perceived latency through more efficient producer-consumer ring buffer between the VM and Domain 0.

VI. Related Work

Prior work on secure logging had focused primarily on non-virtualized settings where the logs are located on storage that is controlled by an insecure machine. [11] describes a cryptographic method for making all log entries generated prior to the logging machine's compromise impossible for the attacker to read. However the attacker can still modify or destroy the logs on the compromised machine, although at the risk of being detected. [12] proposes modifying the device driver to enforce append-only storage semantics. However, since device driver itself is under the control of system being logged, it is vulnerable to being bypassed by the attacker who can then directly write to the raw storage device and modify the logs. Alternative solution of modifying the storage firmware to enforce append-only semantics is too expensive to be practical and vulnerable to firmware modifications. [13] proposes to record logs in write-once media such as optical devices, which tend to be slow and not always available in cluster-based rackmount servers. Another common approach is to maintain a central log server that collects data from all machines over the network. However, the latency of such network transfers is considerably higher than the proposed VAS mechanism. Additionally such network logging techniques may not be readily extensible to individual applications in the manner that VAS logging is extensible. If so desired, the VAS back-end in our architecture can be modified quite easily to transfer the log entries over the network to another machine.

It is also possible to back up traditional logs periodically, but periodic backups incur significantly larger delays between commits and may not capture the point of system intrusion by the attacker.

In virtualized settings, Revirt [2] checkpoints the VM's memory image continuously and replays the checkpoints after a system compromise is detected. However, the logs are system-wide and very low-level, do not capture application semantics, require large disk space, and replays for tracking intrusion events are expensive.

In the domain of intrusion detection and prevention, a number of techniques [5], [8], [14], [3] aim to monitor, detect, and prevent kernel rootkit attacks in VM environments. These techniques primarily monitor the memory-resident kernel code and data of the virtual machine and do not protect the virtual storage device on which logs are stored. Our proposed VAS approach is complementary to such intrusion detection systems and can be used to provide an added level of security by virtualizing the logging storage device.

VII. Conclusions and Future Work

In this paper, we proposed a new virtual storage abstraction for virtual machines called *Virtual Append-only Storage* (VAS) to secure and preserve all system and application logs in a VM. Our VAS-based logging mechanism is completely transparent to applications and the guest OS. It can prevent an intruder from deleting or modifying past logs even after the root privileges of a VM are compromised. Our current VAS prototype can provide secure logging behavior with very low application overhead (11.6% for Apache webs server). We believe that VAS-based secure logging is an instantiation of a more general concept of untamperable virtual storage systems in which all write operations can be converted into log-structured append operations at the level of virtual block device providing the capability of post-intrusion full replay and analysis.

Acknowledgement

This work is supported in part by the National Science Foundation under grants CNS0855204 and CNS0845832.

References

[1] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2008, pp. 51 – 62.

[2] G. Dunlap, S. T. King, M. A. B. S. Cinar, and P. M. Chen, "Revirt: Enabling intrusion analysis through virtual-machine logging and replay," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[3] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proc. Network and Distributed Systems Security Symposium, pages*, 2003, pp. 191 – 206.

[4] A. Joshi, S. T. King, G. W. Dunlap, and P. M. Chen, "Detecting past and present intrusions through vulnerability-specific predicates," in *Proceedings of the 20th ACM symposium on Operating systems principles*, 2005, pp. 91–104.

[5] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "Secvisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity oses," in *Proceedings of 21st ACM SIGOPS symposium on Operating Systems Principles*, 2007, pp. 335–350.

[6] B. D. Payne, M. Carbone, M. Sharif, and W. Lee, "Lares: An architecture for secure active monitoring using virtualization," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, 2008, pp. 233 – 247.

[7] K. Kourai and S. Chiba, "Hyperspector: Virtual distributed monitoring environments for secure intrusion detection," in *The 1st ACM/USENIX International Conference on Virtual Execution Environments*, 2005, pp. 197 – 207.

[8] R. Riley, X. Jiang, and D. Xu, "Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing," in *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, 2008, pp. 1 – 20.

[9] *Intel Trusted Execution Technology*, http://www.intel.com/technology/security/.

[10] *Intel Trusted Platform Module Quick Reference Guide*, ftp://download.intel.com/support/motherboards/desktop/sb/d4858602.pdf.

[11] B. Schneier and J. Kelsey, "Cryptographic support for secure logs on untrusted machines," in *Seventh USENIX Security Symposium*, 1998, pp. 53–62.

[12] Y. Wang and Y. Zheng, "Fast and secure append-only storage with infinite capacity," in *The 2nd IEEE Security in Storage Workshop*, 2003.

[13] R. Finlayson and D. Cheriton, "Log files: An extended file service exploiting write-once storage," *ACM SIGOPS Operating Systems Review*, vol. 21(5), pp. 139 – 148, 1987.

[14] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports, "Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.