Virtual Disk Integrity in Real Time

Chris Rogers
SUNY Binghamton
Binghamton, NY, USA

JP Blake*
Assured Information Security
Rome, NY, USA

***Abstract*—This paper introduces the Virtual Disk Integrity in Real Time (vDIRT) monitor, a mechanism to measure virtual hard disks in real time from the Dom0 trusted computing base. vDIRT is an improvement over traditional methods for auditing file integrity which rely on a service in a potentially compromised host. It also overcomes the limitations of existing methods for assuring disk integrity that are coarse grained and do not scale to large disks. vDIRT is a capability to measure disk reads and writes in real time, allowing for fine grained tracking of sectors within files, as well as the overall disk. The vDIRT implementation and its impact on performance is discussed to show that disk operation monitoring from Dom0 is practical.**

I. Introduction

Computing environments with high integrity requirements continue to invest in trusted computing elements that give insight into the state of the system. Virtualization on client and server platforms increases the complex challenges of understanding system state. However, virtualization also gives unique capabilities such as memory introspection [1], and disaggregration of driver domains from Dom0 [2] that can improve the overall security posture of a system. vDIRT capitalizes on guest disk handling in Dom0 to minimize the performance impact on running VMs while offering insight into disk operations.

II. Related Work

Traditional methods for assessing file integrity rely on a service inside a host operating system that is vulnerable to compromise. Other existing methods for assuring disk integrity are coarse grained and do not scale past small disks.

A. Tripwire

Tripwire [3] is a file integrity monitoring tool that executes inside a host environment. By establishing a baseline database of the entire filesystem, it tracks changes to existing files and the creation of new files. It does not actively prevent these changes; rather, it passively informs an administrator that changes have occurred and provides the option to add those changes in the baseline database. Tripwire report generation does not happen in real time. It is either run manually or periodically as a cron job.

*Corresponding author: blakej at ainfosec dot com

B. XenClient XT

XenClient XT (XCXT) is a security conscious client platform using the Xen [4] hypervisor that takes a first step towards assuring disk integrity of virtual hard disks (VHDs). XCXT uses its toolstack to assure that privileged virtual machines, or *service VMs*, are not modified between boots. XCXT achieves this by computing the SHA1 sum of a VHD immediately preceding VM start, which is compared to the preexisting known good measurement. This method is adequate for smaller VHDs, specifically XCXT's OpenEmbedded based *service VMs*. However, this method does not scale for larger disks (greater than 1GB) as it significantly increases VM boot time.

III. The vDIRT Monitor

A. Background

The blktap2 library is Xen's disk I/O interface. The blktap2 backend establishes itself in the Dom0 kernel and uses an event channel and shared memory ring to communicate with its frontend (blkfront) in a DomU kernel. I/O requests from the guest VM userspace pass into the guest's kernelspace where they are redirected to Dom0 and blktap. Once the I/O requests reach the Dom0 kernel, they are sent to Dom0 userspace and the tapdisk utility. Tapdisk determines which disk I/O library should be used to handle the operation and forwards it appropriately. Blktap2 provides an interface to support custom disk types in addition to the existing *block-aio*, *block-ram*, and *block-vhd* types. Once the read or write is complete, the data is sent back through the shared memory ring to the front end for the DomU to utilize. Figure 1 shows a high level diagram of this architecture.

The VHD format is flexible and based off an open specification. In addition to those advantages, vDIRT uses the VHD format in order to more closely model the XCXT case study. VHDs are created in one of three formats: *fixed*, *dynamic*, or *differencing*. The fixed format allocates a file with the same size as the virtual disk. It consists of a data section followed by a footer containing simple metadata about the disk. The dynamic format is more complex; the file is only as large as the data written to it, and thus requires additional metadata to manage this scheme in the form of a header and block allocation table. The differencing format represents the current state of a VHD by maintaining a grouping of altered blocks with respect to a static parent image. vDIRT currently supports the dynamic VHD format, as it is widely used.

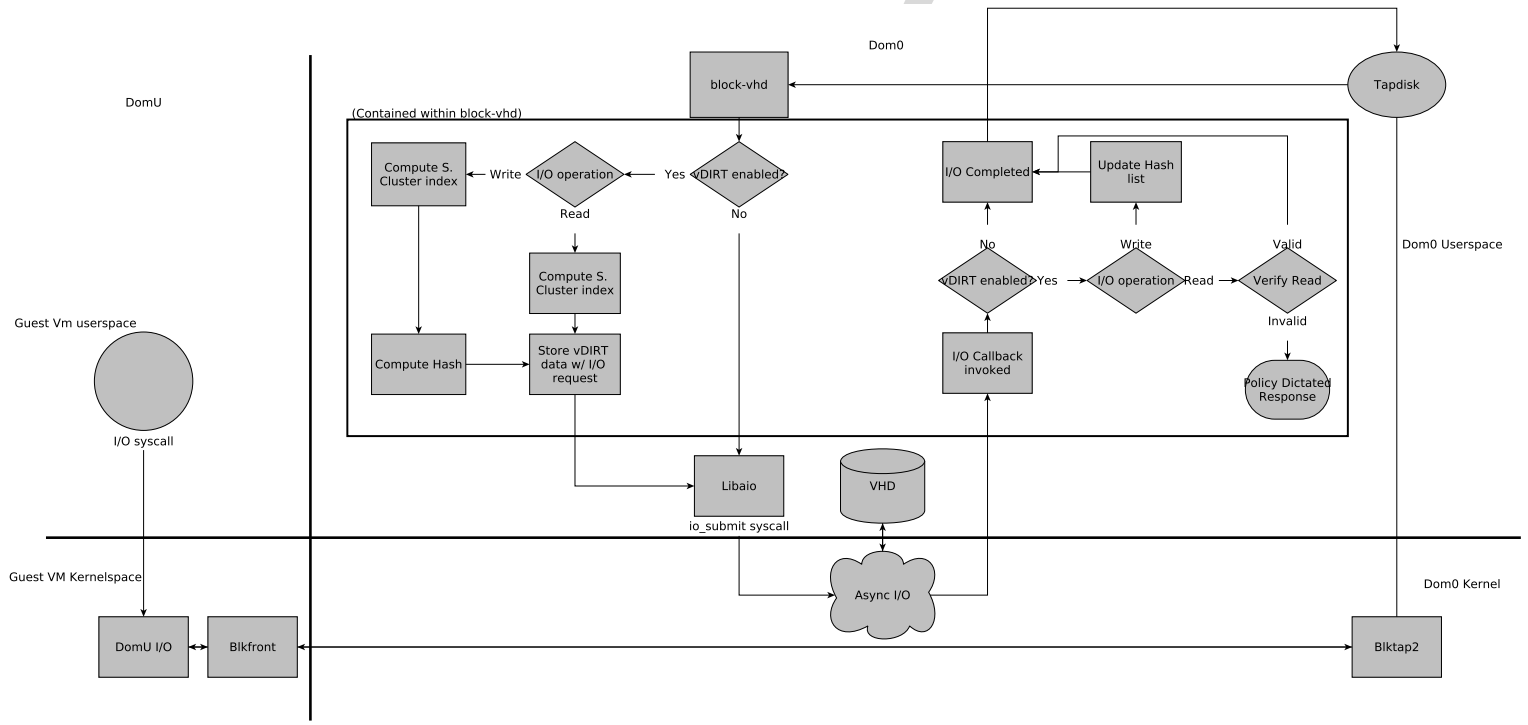

Fig. 1. High level diagram of the flow of I/O operations from the Guest VM to Dom0 blktap and vDIRT measurements.

B. Implementation

To amortize the performance penalty of measuring large VHDs the VHD metadata structure was modified to include support for the hashing scheme. Specifically, a *hash header* is introduced after the mirrored footer containing the following information:

- Absolute byte offset to the next data structure (disk header).
- The number of hash entries in the header.
- A flag to indicate measurements are active.
- The hashes of the disk's *sector clusters*. A *sector cluster* is a group of eight 512-byte sectors on disk, the granularity at which Xen generally executes data I/O operations.
- Padding to ensure the hash header is 512-byte aligned to conform to O DIRECT specifications for reading/writing.

Unlike other disk metadata, the size of the hash header cannot be determined statically due to its dependence on the size of the entire VHD. To avoid the performance overhead of maintaining a dynamically sized list of hashes in memory, sufficient disk space is allocated to hold hashes of all the sector clusters. This method is also necessary to avoid realigning disk data and metadata following a new block allocation. Unallocated sector clusters are given a NULL value as their initial hash. These optimizations are handled at VHD creation time via a modified vhd-util toolstack.

The hash header also has an accompanying implementation in memory because it is not practical to perform an additional disk read or write for each I/O operation submitted. After the VHD has been opened by blktap, all hashes are cached in memory to avoid the slowdown that would be caused by disk I/O. Efficient memory usage becomes a concern with large VHDs. However, this implementation only requires 400MB of memory to maintain all the hashes for an 80GB VHD. Reducing memory usage for larger VHDs is a focus for future work. As a result, accessing the hash of a given sector cluster

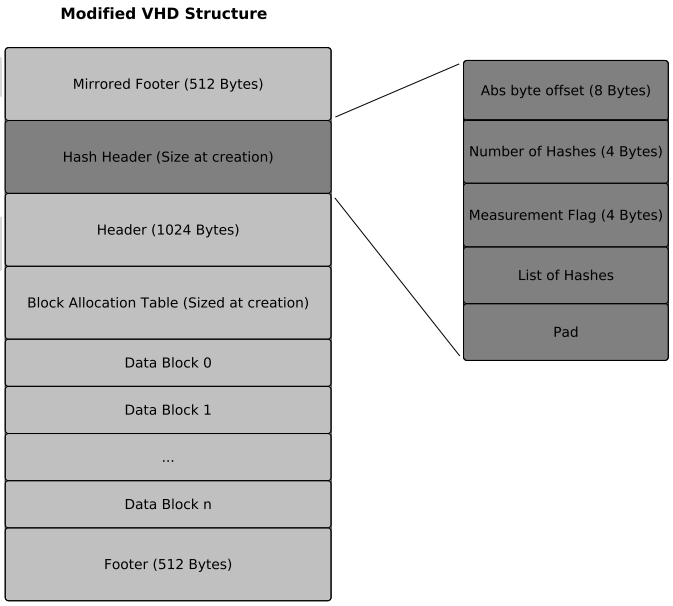

Fig. 2. Diagram of the modified VHD structure.

can be completed in constant time. This is ideal given the many I/O operations performed during the lifetime of a VM, and that real time measurements should have an imperceptible impact on runtime performance.

Once the VHD has been created, any future reads and writes through the tapdisk utility will be measured. When a disk write is scheduled, it is dispatched through block-vhd where the operation is determined to be a data write. Then, using global and block offsets, the target sector cluster index is calculated with respect to the entire VHD. Finally, the data in the buffer to be written is SHA1 hashed and these two values are stored inside the I/O request structure. When verifying a data read, the sector cluster index is also computed, but no hashing is performed since the data has not yet been read into memory. Instead, space is allocated in the I/O request to store the index that will be accessed on the return of the callback.

Upon completion of an asynchronous I/O operation, the callback function vhd-complete is invoked. If the completed operation was a data write, the list of hashes in memory is updated with the precomputed hash. Otherwise, if the completed operation was a data read, the data is hashed and its sum is verified against the tracked value. Non-matching hashes indicate an invalidation of disk integrity during an offline state. It is possible for policy to dictate the corresponding action in this event, such as to halt operations or warn the user. Figure 1 provides a high level illustration of this implementation.

To match XCXT's disk measurement scheme, a single, unified measure of the VHD is obtained by SHA256 hashing the list of sector cluster hashes. Vhd-util was modified to support this operation.

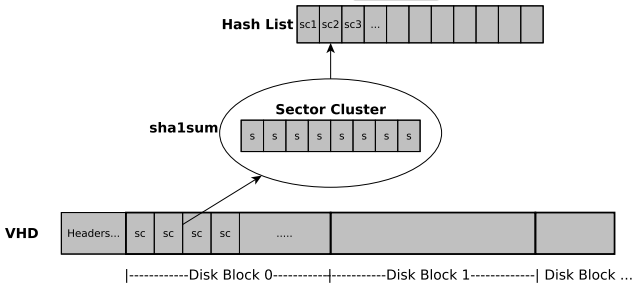

Fig. 3. Illustration of hash tracking scheme. *sc* is a *sector cluster* and *s* is a *sector*.

C. Protecting vDIRT

As a modification to blktap2, vDIRT is within the Dom0 trusted computing base. Standard strategies to ensure the integrity of Dom0 should be employed. The system should comprise of a measured launch using TBOOT and Intel Trusted Execution Technology, coupled with sealed storage of the encryption keys belonging to a protected partition. In order to guarantee the integrity and consistency of the hash header in an offline state, it is encrypted using AES-CBC 256 before writing changes back to disk. Its encryption key is stored on a protected partition within Dom0. This way, tampering with the hash header will result in failed decryption, indicating a breach of disk integrity.

In an effort to maximize performance, vDIRT is designed to commit tracked hashes to disk only on VM shutdown. In the context of power interruptions, this becomes a concern. If the interruption was malicious in nature, such as an attempt to modify data and circumvent read verification, vDIRT will still detect invalid data the next time it performs a read since the sector clusters on disk and the tracked hashes will be out of sync. However, if the power interruption was coincidental, the VHD should not necessarily be marked as compromised. To help distinguish between the two cases, a simple flag that indicates a proper hash commit could be tracked.

D. vDIRT Performance

The testing environment for vDIRT used a Dell Optiplex 980 with an Intel quad core i5 CPU @ 3.33GHz, 8GB DDR3 RAM, one 160GB 7200RPM Western Digital hard drive and one Chronos SATA 3 120GB SSD. Xen, Dom0, and all other applications were located on the HDD, and all VHDs involved with testing were located on the SSD to ensure that VM I/O operations were better isolated from normal system overhead.

Figure 4 shows the average I/O operations per second (IOPS) of three different types of synchronous I/O performed from inside a VM using the *fio* program. The *Modified* label indicates blktap was running with vDIRT measurements active, while *Native* indicates it was running without any modifications. Writes were configured to perform fsync after every eight write operations and at the completion of a job to force a high frequency of vDIRT measurements. This helps to better illustrate the performance relationship between the two implementations. Figure 5 shows the average bandwidth of the same synchronous operations under the same conditions.

As expected, vDIRT is slightly less efficient with I/O than native blktap due to the extra work performed during runtime. However, these tests illustrate that there is no large disparity in VM runtime performance between the two implementations. It is also worthy to note that there was no discernable difference in Dom0 CPU usage while vDIRT was active.

Finally, Figure 6 shows the difference in runtime of the current XCXT VHD measurement scheme (sha1sum) and vDIRT's scheme (vhd-util gethash). By design, vDIRT's scheme reduces the amount of data that must be hashed by a factor of over 200. The example below shows how a 1GB VHD would have a sector cluster hash list of 5MB.

$$1\,sector\,cluster\;=\;8\,blocks\,*\,512B\,per\,block\;=\;4KB \quad (1)$$

$$1GB\;/\;4KB\,per\,sector\,cluster\,(SC)\;=\;262144\,SC \quad (2)$$

$$262144\,SC\;*\;20B\,per\,SC\,SHA1\,hash\;=\;5MB \quad (3)$$

The data shows that vDIRT offers a significant reduction in disk measurement overhead while having a minimal impact on runtime operation.

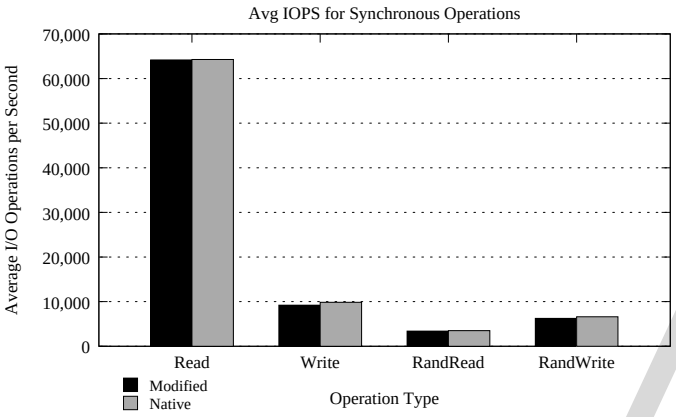

Fig. 4. Average IOPS of several synchronous I/O tasks.

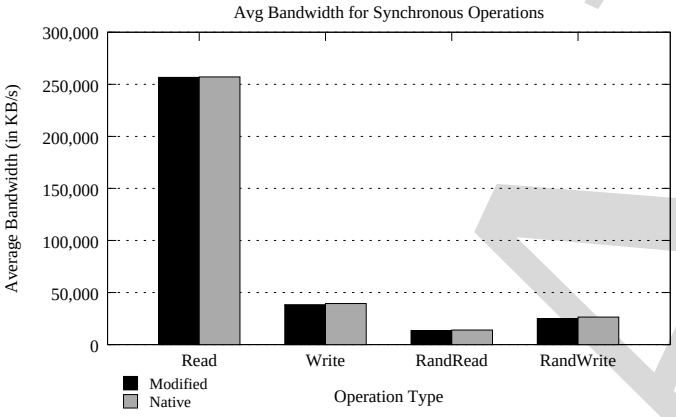

Fig. 5. Average bandwidth of several synchronous I/O tasks.

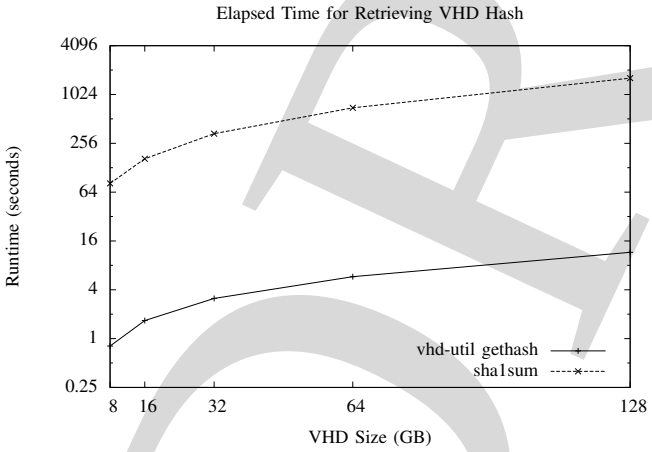

Fig. 6. Elapsed time to retrieve the VHD hash.

IV. Future Directions

Work is ongoing in vDIRT to further improve the tool with respect to speed and memory usage. The minimal difference in performance between native blktap and blktap with vDIRT can be attributed in part to the constant time access of the tracked sector cluster hashes. However, as this strategy requires all hashes to be in memory at once, it quickly consumes a vast amount of space as VHD size increases above 128GB. In systems where memory is limited or multiple VMs are running simultaneously, this could be detrimental to performance. A possible solution may be to implement a caching scheme to reduce memory usage. Challenges here include choosing an optimal replacement strategy and cache size to mitigate thrashing and the overhead of additional disk reads and writes. The caching scheme would contribute extra overhead and complexity, but the space/performance tradeoff can be reevaluated to reach an efficient balance.

vDIRT has possible applications ranging from file integrity monitoring and disk forensics to malware analysis and protecting files on disk. To more closely approximate Tripwire functionality, additional work is necessary to be filesystem type aware. Understanding the filesystem context is important in tracking file inode changes where a files logical block address offsets are changed. vDIRT offers a useful capability in supporting honeypot VMs where all disk reads and writes could be logged and later replayed to support the analysis of an attack. Furthermore, vDIRT could be modified to support a policy engine that defines files that should not be altered on disk. For example, in VMs using SELinux, it is crucial that the GRUB configuration file not be altered to turn off SELinux.

V. Conclusion

vDIRT is a significant improvement in capability over legacy solutions for disk measurement and file integrity assurance. It can be used in conjunction with current VM memory introspection solutions to improve on overall understanding of system operations. vDIRT offers a 200 fold improvement over existing solutions when queried for a single, unified measurement of a VHD.

References

[1] P. A. Loscocco, P. W. Wilson, J. A. Pendergrass, and C. D. McDonell, "Linux kernel integrity measurement using contextual inspection," in *Proceedings of the 2007 ACM workshop on Scalable trusted computing*. ACM, 2007, pp. 21–29.

[2] P. Colp, M. Nanavati, J. Zhu, W. Aiello, G. Coker, T. Deegan, P. Loscocco, and A. Warfield, "Breaking up is hard to do: security and functionality in a commodity hypervisor," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 189–202.

[3] G. H. Kim and E. H. Spafford, "The design and implementation of tripwire: A file system integrity checker," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*. ACM, 1994, pp. 18–29.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.