Towards Disk-Level Malware Detection

Nathanael Paul Sudhanva Gurumurthi David Evans
University of Virginia, Department of Computer Science
Charlottesville, VA
{nate, gurumurthi, evans}@cs.virginia.edu

Abstract

Disk drive capabilities and processing power are steadily increasing, and this power gives us the possibility of using disks as data processing devices rather than merely for data transfers. In the area of malicious code (malware) detection, anti-virus (AV) engines are slow and have trouble correctly identifying many types of malware. Our goal is to help make malware detection more reliable and more efficient by using the disk drive's processor. Using the extra processing power available on modern disk drives can provide significant advantages in detecting malware including reducing the traditional AV engine's workload on the host CPU by partitioning the workload between the host AV engine and the disk drive, improving the detection of stealth malware by providing a low-level view of the system, and recognizing virus behavior by observing disk I/O traffic directly. Several research questions must be addressed before these benefits can be realized: how to correctly partition work between the AV engine and the disk drive processor, how to design interfaces between the operating system (OS) or host AV engine and the disk drive that provide satisfactory performance without compromising security, and how to recognize malicious behavior based on the dynamic analysis of low-level data accesses.

Keywords: dynamic analysis, malware detection, virus detection, disk drive processor.

1. Introduction

Malware detectors face three major challenges: they must have false positive rates very close to zero, they must have minimal performance overhead, and they must be able to detect a large number of known viruses and an unlimited number of possible variants. Moving malware detection and response to the disk drive processor offers promising opportunities on all three of these problems since the disk can analyze all I/O traffic with little overhead. Current methods of virus detection can have an overhead potentially as high as 129% [21]. In fact, one company is marketing a hardware coprocessor explicitly for virus detection to lessen AV overhead by using regular expressions along with other techniques [20], and Symantec has a patent on a device (or software) that can be queried to match a regular expression on data and block that data from storage if there is a match [7]. These hardware solutions present a bottleneck, since higher disk activity will require higher loads on a centralized hardware coprocessor. Our goals are to speedup and improve the accuracy of virus detection without requiring specialized hardware.

Recent trends are making storage devices more aware of application needs (application-aware) and making devices actively perform application-specific code on the storage system itself (active storage) [4, 18]. For instance, rather than doing encryption in software, Seagate recently announced a drive that can perform hardware-based encryption [16]. A drive's processor is about three generations behind a high-end modern CPU [1], and it is currently underutilized. Sivathanu et al. model a semantically-smart disk using a Pentium III 550 Mhz processor, and this trend of increasing processing power should continue as Application-Specific Integrated Circuit (ASIC) technology continues to provide higher speeds within disk power constraints [18]. We have the opportunity to have work done in the disk processor nearly for free. A moderate amount of extra computation on the hard drive CPU scales better than a larger I/O load, since we can hide the processing with the physical data transfers that will incur longer mechanical delays. In using the disk processor to aid malware detection, the I/O is already transferred by the disk, so the only cost is what we incur by also examining the request and data.

The next three sections explore areas in malware detection that significantly benefit from additional processing power. Section 2 deals with the issues in partitioning the AV engine workload between the primary CPU and the disk drive. If the AV engine can let an active disk perform work on its behalf while freeing up the host CPU to service application tasks, then this will increase overall system throughput. Section 3 discusses augmenting low-level protection of machines from rootkits. Many rootkit detection tools depend on a true low-level view of the system, and the closest component of the non-volatile state of the system is the disk drive. Our last example discusses the dynamic analysis of disk I/O.

2. Partitioning Workload

Some of the main actions that an AV engine performs are heuristic scanning, signature matching, and emulation [19]. Heuristic scanning is used to quickly identify traits of a program to act as a filter to more expensive scanning techniques (like emulation), but the most dominant action is simple string matching. Emulation is also an expensive action, and the AV software must be selective on the programs it chooses to emulate and how long it performs emulation. On a single processor system, the AV engine overhead can cost up to 129% [21]. Although the disk drive processor is a suitable place to offload some of the burden of an AV engine, we must first address several issues including: partitioning the workload between the AV engine and the disk processor, minimizing I/O delay, and the interaction between the AV and the disk.

Partitioning malware scanning is especially important if we can exploit parallelism for the scanning process. For this design, we envision an active and application-aware disk where the disk acts independently of the host AV engine.

Since we are using a disk processor, we are limited to a small cache, typically 8–16MB. This limits the disk scanning to use a smaller signature database. The size of the Symantec AV update executable for October 8, 2005 is 8.86 MB [12], but our signatures will be at the lower level of disk blocks. We face the challenge of creating small signatures with low false positive rates. While using the disk cache for signatures, we will also need to study the trade-off of the OS using the disk block cache.

Some polymorphic malware may be difficult to identify, since encryption can be used before writing to the disk. Although knowing the content of I/O traffic can be more useful, the disk may still identify malware according to the location of a written block even if the block is encrypted. Some malware can be recognized through simple string matching, enabling the disk to recognize the malware whenever a known string is written to the disk. Of course this would not work for encrypted I/O, so a signature for encrypted I/O could capture the ways in which the PE file was being manipulated. For instance, one questionable PE file modification is overwriting the entry point of execution. We must be careful to keep the false positive rate low if we use these types of heuristic defenses.

These designs warrant further study into how to minimize I/O delay from regular disk requests and how to set up the communication channel between the disk and the OS or AV engine. We cannot overload the disk processor to the point where regular I/O suffers, but we will harness the idle processing power currently available. The second problem of establishing a secure line of communication is not possible with current PC design, but there are some things we can do to raise the bar for a successful system compromise discussed in the following section.

3. Dynamic Analysis of I/O Requests

In this application area, we use the drive in a more active capacity to scan the disk traffic for malicious activity in conjunction with the host AV engine. Higher-level detection code may not easily detect the malicious behavior, but the disk may see malicious I/O and stop it before anything bad happens.

For example, the W32/Funlove virus infects local and networked drives by adding a small amount of data ("Fun Loving Criminal") to the end of each PE file [6]. As the virus enumerates the network shares, it writes to remote PE files through memory mapped file I/O. On-access virus scanners work by catching certain file events like open, close, and create, and they could not detect the virus without more information. Funlove used memory-mapped I/O to compromise remote machines, so traditional scanners could not recognize the virus until it had already infected the disk [19]. Szor goes on to recommend other defense mechanisms including behavior blocking and a network IDS, but an active disk scanning for anomalous traffic (e.g., malicious traffic that installs itself as a Windows service) can prevent intrusions like Funlove with much less cost than an IDS.

One consequence of scanning I/O patterns at the disk-level is that we lose semantic information that we would otherwise have in a typical AV engine. The disk will only receive requests that read or write blocks at a given location, but the disk needs to know what these blocks map to in order to do something useful. One way in which we could bootstrap the disk is by providing a mapping of disk blocks to applications at OS installation. Without talking to the OS, we will need to provide all necessary semantic information about the file system at the OS installation. This comes at a cost – not being able to securely update the disk software after installation. If we need a more extensible design, then we can allow interaction between the AV and disk drive, but we run the risk of having these interfaces compromised.

Some current, and most future PCs, are likely to support a Trusted Platform Module (TPM) that enables a secure bootstrapping process. This has benefits in preventing virus propagation, but as long as users are able to install additional software, TPMs will not stop all viruses. However, we can use a TPM like Intel's LaGrande for secure communication between the AV and the disk [10]. If the AV engine has a trusted part of code protected by hardware, then the AV engine can attest the calls the disk makes and notify the user to take additional actions if necessary.

4. Detecting Rootkits

Rootkits are a form of malware that are installed by an attacker to keep stealth or secretive access to a machine. This is often accomplished by altering some part of the OS [9]. Rootkit detection tools like Strider Ghostbuster [22], RootkitRevealer [3], and Blacklight [5] perform a high-level scan and a low-level scan of a machine looking for a discrepancy between the scan reports. The high-level scan will use the Windows API or a command like "dir /s /b", and the low-level scan will read the Master File Table (MFT), raw hive files (Windows registry), and the kernel process list.

The security of these mechanisms relies on the difficulty of implementing code that could intercept these low-level reads and construct false MFT, hive, or process data. While this may give detection tools the ability to detect most current rootkits, it is only a matter of time until rootkits are developed that can fool the low-level scans. The rootkit detector will call some API to read the low-level data, and this can always be hooked. The detector could implement the functionality of a disk device driver itself to communicate directly to the hardware, but some rootkits are now unhiding files when they detect a rootkit detector performing a file system scan, so the rootkits remain undectected while the unhidden files do not get reported as a difference between the two scans [15]. This battle between rootkits and detection tools will inevitably continue.

To setup the disk to perform a low-level scan, the disk must have more information about the disk blocks. We bootstrap this information at the OS installation, and the disk then has the associations for registry data and the MFT. The kernel process information is kept in memory at run-time, so this information is not accessible from the disk. Since the disk will be performing the file system scan, the scan has the advantage of remaining undetected from any rootkit.

To recover from a low-level malware infection, we can set up the disk to protect certain disk blocks associated with core OS files again specified at OS installation time. Although the OS may have system restore data (e.g., MS Windows), these blocks would not be writable to any code outside of the disk drive. The main problem is performing an update to a protected OS file. Assuming no rootkit has compromised the machine, the user could download an update and override the protection mechanism to apply the update. Of course, this does force the user to trust the OS to not have been compromised. Any time one of the protected blocks is overwritten, the disk can make a backup to some portion of the disk that is not accessible to the OS, in case restoration is needed later. If the disk does indeed find a discrepancy between the high and low level scans, then the latest known clean block can be restored, and we can perform another system scan to make sure all traces of the malware are removed.

The advantages of this approach far outweigh the overhead of storing a few KB/MB on the disk. Capacity is not at a premium, since we currently have consumer disks reaching as high as one-half TB now [8]. The main weaknesses in this approach are updates of registry data and communication between the AV engine and the disk. When the user installs an application it can potentially destroy the integrity of the registry, but the disk may be able to detect a malicious update to the registry in some cases. For the updates that are not deemed malicious, we must either depend on the recovery mechanism in case it is later identified as malicious or allow the disk to receive updates about the protected disk blocks from the OS.

To compare the high-level and low-level scan, the OS will need the low-level scan information, or the disk will need high-level scan information. Because the design requires comparing the results from the OS and disk, the communication link is again a vulnerable target. As suggested in the previous section, we can make use of a TPM. If a TPM is not available, then we have at least raised the bar of system compromise.

Another example of low-level AV software being circumvented by rootkits is within a filesystem filter driver [9]. To process an I/O request packet (IRP) in Windows, the IRP is passed through a chain of filter drivers before reaching the lowest-level device driver [11]. Many AV engines install their own filter driver in this chain to process incoming IRPs, but even these filters could be circumvented. Using the disk for scanning ensures that malicious traffic can be scanned while the scan itself cannot be circumvented.

5. DADDIO

We are building a new tool to *D*ynamically *A*nalyze *D*isk *D*rive *I/O* (DADDIO) while offloading the CPU workload and aiding in low-level malware detection. DADDIO can provide interfaces to the AV engine to perform string matching and for viewing the low-level filesystem details, and it will analyze disk I/O for malicious activity. If the AV uses software interfaces to DADDIO, then we must use a TPM to use DADDIO securely.

We may be able to leverage DADDIO without a TPM, but we will lose the capability to communicate securely with the host OS. To perform services on behalf of the host AV engine, DADDIO will throttle its own execution workload if the I/O performance suffers. DADDIO's other main action of scanning for malicious disk I/O will be performed during each write to the disk. Reads do not matter if we assume no malicious blocks exist on the disk before DADDIO is activated and DADDIO can prevent malicious writes to the disk.

Recovery from detected malicious I/O traffic can be done without interaction from the AV engine. Without communication with the host OS, we do not risk compromise of the communication channel, but DADDIO has no way of indicating a problem to the user. At the worst, DADDIO can simply suspend all disk I/O. Once users observe the system has frozen from the suspended disk, they will most likely perform a reboot, erasing the malware from the system. Note that this eradicates the malware, since DADDIO prevented it from ever writing to the disk. If the virus activity can be isolated, DADDIO can continue to service regular disk I/O while denying disk access to the malicious process performing I/O. One possibility is to adopt the failure-oblivious computing approach introduced by Rinard et al. [14], and simply write different data on the drive than the malicious code. This approach has proven effective in masking memory errors to keep a faulty server running [14]. Our aim is to allow the OS to safely continue execution without malicious corruption.

6. Related Work

Others have also studied the idea of providing AV services outside of the host machine AV engine. Work by Pennington et al. studied an AV implementation on an NFS server [13]. In hardware, Silberstein [17], Tarari [20], Symantec [7] have proposed ideas to offload AV computation from the main host.

Silberstein observes the signature matching overhead for the open-source AV tool, ClamAV [2], can be as high as 40%. He suggests using a hardware coprocessor to assist in string matching.

Pennington's implementation of an IDS uses an NFS server to support a rule-based IDS system [13]. For their IDS, they require a filesystem separate from the host, so they can guarantee protection for the IDS administration system even if the host has been compromised. However, this protection does not extend to the client, so a compromised client (e.g., trojan) could allow unauthorized access to data on the NFS server [Gobioff99]. Our design protects the local machine by placing an AV engine directly on the disk drive that can partition the workload between the main AV engine and the disk drive AV engine.

The Tarari [20] and Symantec [7] implementations describe designs that can also be used as a hardware or networked offloading engine, respectively. Both designs are for offloading the workload of AV engines, but neither one provides for offloading on the local machine.

One critical difference is that our design can capitalize on the disk already being on the critical path of the data, so we are now analyzing the data rather than just performing data transfers. We avoid the potential bottlenecks of having to pass data through dedicated (and more expensive) hardware solutions.

7. Conclusion

Active and application-aware disks can be useful in malware detection by offloading part of the AV's offload, providing a closer view of the low-level filesystem, and scanning for malicious disk access patterns using low-level disk I/O. We advocate using the disk drive processor for malware detection, and we have presented three motivating examples showing the advantages of using the disk processor for detecting viruses and rootkits. Current virus scanning can be expensive (e.g., emulation and string scanning), and the virus scanner must minimize its own overhead. Lowering the host AV scan-time frees up more processing time for more analysis or for performing other application tasks. If we can capitalize on detailed dynamic analysis of disk I/O, we may be able to avoid more viruses like W32/Funlove. We are currently investigating ways to partition the load between the host CPU and the disk drive CPU and are identifying the best techniques that can benefit from low-level disk block information.

Acknowledgements

The authors thank Peter Szor for helpful discussions regarding this work. This work was supported in part by grants from the National Science Foundation (NSF CAREER CCR-0092945 and NSF ITR EIA-0205327).

References

[1] Anurag Acharya, Mustafa Uysal, and Joel Saltz. Active Disks. *Eigth Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), October 1998.

[2] Clam AntiVirus. http://www.clamav.net/.

[3] Bryce Cogswell and Mark Russinovich. *SysInternals RootkitRevealer*. http://www.sysinternals.com/utilities/rootkitrevealer.html.

[4] Information Storage Industry Consortium. *Data Storage and Systems (DS2) Roadmap*. January 2005.

[5] F-Secure. Blacklight. http://www.f-secure.com/blacklight/.

[6] F-Secure Virus Descriptions: FunLove. http://www.f-secure.com/v-descs/funlove.shtml.

[7] John K. Hile, Matthew H. Gray, and Donald L. Wakelin, *In Transit Detection of Computer Virus with Safeguard*. US Patent 5319776.

[8] Hitachi Deskstar 7K500. http://www.hitachigst.com/portal/site/en/menuitem.8f07a3c3d3a7a12d92b86b31bac4f0a0/.

[9] Greg Hoglund and James Butler. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley, 2005.

[10] Intel Corporation. *Lagrande Technology Architectural Overview*. September 2003. http://www.intel.com/technology/security/downloads/LT_Arch_Overview.pdf.

[11] Rajeev Nagar. *Windows NT File System Internals*. O'Reilly, September 1997.

[12] Norton AntiVirus Virus Definitions. October 8, 2005. http://definitions.symantec.com/defs/20051008-002-i32.exe.

[13] Adam G. Pennington, John D. Strunk, John Linwood Griffin, Craig A.N. Soules, Garth R. Goodson, and Gregory R. Granger. Storage-based Intrusion Detection: Watching Storage Activity for Suspicious Behavior. *12th USENIX Security Symposium*. Washington D.C. August 2003.

[14] Martin Rinard, Cristian Cadar, Daniel Dumitran, Daniel M. Roy, Tudor Leu, and William S. Beebee, Jr. Enhancing Server Availability and Security Through Failure-Oblivious Computing. *Sixth Symposium on Operating Systems Design and Implementation* (OSDI), December 2004.

[15] Joanna Rutkowska. *Thoughts about Cross-view based Rootkit Detection*. June 2005. http://invisiblethings.org/papers/crossview_detection_thoughts.pdf.

[16] Seagate Press Release. *Hardware-based Full Disc Encryption Security*. June 8, 2005. http://www.seagate.com/cda/newsinfo/newsroom/releases/article/0,,2732,00.html.

[17] Mark Silberstein. *Designing a CAM-based Coprocessor for Boosting Performance of Antivirus Software*. March 2004. http://www.technion.ac.il/~marks/docs/AntivirusReport_revised_version.pdf.

[18] Muthian Sivathanu, Vijayan Prabhakaran, Florentina Popovici, Timothy Denehy, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Semantically-Smart Disk Systems. *Second USENIX Conference on File and Storage Technologies* (FAST), March 2003.

[19] Peter Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley, 2005.

[20] Tarari. *Anti-virus Content Processor*. http://www.tarari.com/antivirus/index.html.

[21] Derek Uluski, Micha Moffie, and David Kaeli. Characterizing Antivirus Workload Execution. *Workshop on Architectural Support for Security and Anti-Virus* (WASSA). October 2004.

[22] Yi-Min Wang, Doug Beck, Binh Vo, Roussi Roussev, and Chad Verbowski. *Detecting Stealth Software with Strider Ghostbuster*. MSR-TR-2005-25.