SystemWall: An Isolated Firewall using Hardware-based Memory Introspection

Sebastian Biedermann[1] and Jakub Szefer[2]

[1] Security Engineering Group
Department of Computer Science
Technische Universität Darmstadt
biedermann@seceng.informatik.tu-darmstadt.de

[2] Computer Architecture and Security Laboratory
Department of Electrical Engineering
Yale University
jakub.szefer@yale.edu

Abstract. Memory introspection can be a powerful tool for analyzing contents of a system's memory for any malicious code. Current approaches based on memory introspection have focused on Virtual Machines and using a privileged software entity, such as a hypervisor, to perform the introspection. Such software-based introspection, however, is susceptible to variety of attacks that may compromise the hypervisor and the introspection code. Furthermore, a hypervisor setup is not always wanted. In this work, we present a hardware-based approach to memory introspection. Dedicated hardware is introduced to read and analyze memory of the target system, independent of any hypervisor or OSes running on the system. We apply the new hardware approach to memory introspection to built-up an architecture that uses DMA and fine-grained memory introspection techniques in order to match network connections to the application-layer while being isolated and undetected from the operating system or the hypervisor. We call the proposed architecture SystemWall since it can be a standalone physical device which can be added as an expansion card to the mother board or a dedicated external box. The architecture is transparent and cannot be manipulated or deactivated by potential malware on the target system. We use the SystemWall in the evaluation to analyze the target system for malicious code and prevent unknown (malicious) applications from establishing network connections which can be used to spread viruses, spam or malware and to leak sensitive information.

1 Introduction

Memory introspection is a powerful technique for analyzing code and data contained in memory of a running system. Past approaches have focused on Virtual Machine (VM) based introspection techniques. In this work, we present another type of introspection, based on dedicated hardware components and software that can perform the introspection independent of any software running on the target system.

We apply the introspection techniques to build the SystemWall, a firewall-like system that can analyze memory of the running target computer, detect malicious or unknown applications and block their connections to the external world. This can prevent spread of viruses, malware, spam or even leakage of sensitive documents by malicious, unknown applications. SystemWall is logically fully external to the target computer system: it can be implemented as a stand-alone box that connects to target system or a dedicated extension card on the motherboard.

1.1 Security through Firewalls

In general, firewalls are either an external device only connected to the network or software-based and installed on a target computer. Firewalls control the incoming and outgoing network traffic depending on network events and predefined rules. Firewalls can be simple packet filters blocking or allowing network packets depending on their header information like the source and the destination. Other firewalls can analyze the content of network packets (deep packet inspection), for example with regular expressions [33], which allows the definition of more sophisticated rules.

External firewalls, however, do not have insight into the contents of physical memory of a target system and cannot make decisions based on what code is accessing or handling the network traffic on that system. Software-based firewalls, on the other hand, monitor the network traffic as well as the application-layer of a target computer system and control the incoming and outgoing traffic depending on rules which refer to protocols and states of the involved applications. Software-based firewalls are widely used as personal firewalls. However, software-based firewalls are installed on a target system and can be the target of attacks themselves. In particular, malware can successfully execute attacks against the operating system and can manipulate deployed rules, disable or change the mode of the installed software-based firewall's operation. This way, the user does not even notice the infiltration of the operating system and deems the system to be secured by trusting the running firewall and assuming its correct operation.

External firewalls with added ability to analyze memory of the target, like software-based firewalls, would combine best of both approaches – this is the motivation for SystemWall design.

1.2 Leveraging DMA for Security

DMA (Direct Memory Access) is a specification that allows hardware devices to bypass the Central Processing Unit (CPU) and access the system memory directly. This brings the advantage that the CPU can perform other useful tasks while DMA operations are in progress and it can also accelerate certain tasks. A lot of hardware devices like graphic cards, disk controllers or network cards use DMA.

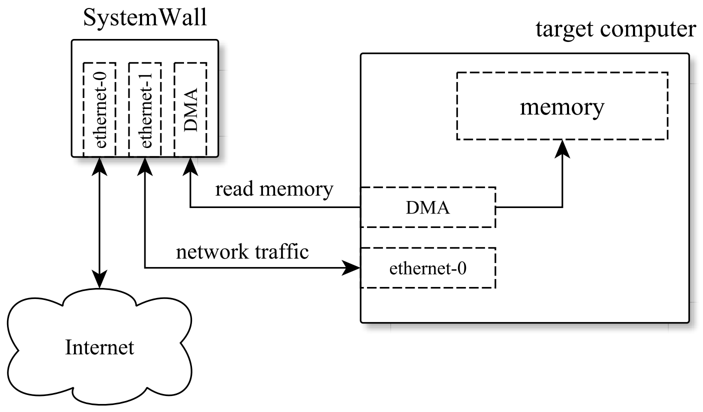

Fig. 1. The SystemWall deployed as an external hardware device which can read the memory of the target and regulate the target's network traffic. Only one ethernet connection is shown for the target in this figure, if target has multiple network interfaces, such as two ethernet ports, SystemWall should regulate each ethernet connection.

DMA has been the focus of security researchers for some years, because it allows to dump the memory of a system through certain external interfaces while bypassing the operating system and any software-based security restrictions. In particular, DMA can be exploited by an attacks on unattended, running systems which provide DMA via an external buses like ExpressCard, FireWire or Thunderbolt to create a dump of the memory. Afterwards, the memory dump can be investigated using forensic techniques in order to retrieve passwords or other sensitive information.

However, DMA can also be used to increase security of a computer system and prevent numerous attacks. In particular, DMA can be used to transparently read the memory contents via the hardware, contents which can later be analyzed for malicious programs or network connections – as we do in SystemWall.

1.3 SystemWall Overview

In this work, we benefit from DMA to setup an isolated firewall-like system which we call SystemWall, shown in Figure 1. The SystemWall can be deployed between the system which it protects and the Internet, and intercept all packets traveling from and to the target system. Placing SystemWall between the Internet and the target system allows for it to, for example, delay network packets going to or from the target while the target's memory is analyzed to validate the packets are related to a legitimate, non-malicious application. To perform the analysis, SystemWall transparently uses DMA and fine-grained memory introspection techniques to match detected initiations of new network connections to applications running on the system. It can use application names, hashes or even scan for shellcode to detect malicious applications and prevent them from making network connections. The SystemWall remains undetected from the operating system and is a combination of a personal software-based firewall and hardware-based memory introspection. This way, the SystemWall cannot be manipulated or disabled by potential malware which could infect the system. Given its access to target's memory, it can monitor the applications and control network connections of these applications to prevent spread of malware, viruses or potentially leakage of sensitive files. Figure 1 shows a block diagram of the SystemWall architecture, fully described in Section 3.

1.4 Paper Organization

The remainder of the paper is organized as follows. Section 2 presents related work. Section 3 explains the architecture and details of the SystemWall implementation. Section 4 evaluates the proposed architecture and Section 5 discusses limitations. Finally, Section 6 concludes.

2 Related Work

This section presents related work in the field of physical memory acquisition with the help of hardware extensions, attacks based on this, countermeasures and methods that use DMA for other purposes. Furthermore, we list some related work in the field of tamper-resistant security architectures that can run isolated from the target system and that are based on hardware extensions or a hypervisor setup.

2.1 Physical Memory Acquisition

DMA has been previously exploited to execute attacks against a running system. Attackers have used buses like USB, FireWire, Thunderbolt or PCMCIA cards to transparently acquire the volatile memory of a running system without being detected and without being the subject to software-based control mechanisms. Afterwards, the memory dump can be analyzed for sensitive data like passwords. However, attackers can also write to the memory pages of the running system and this way modify the system's properties or work-flow on-the-fly. For example, a Windows 7 kernel can be directly manipulated in the memory in order to allow an attacker to log-in using a blank password [2]. Nowadays, standard DMA attacks and further procedures are even implemented in exploitation frameworks [7].

As a consequence of these attacks, several countermeasures have been suggested and are of interest, e.g., [23] or [24]. In particular, hardware-based memory acquisition of specific memory regions can be prevented by modifying the processor's North Bridge's memory map [22]. Also, malware that uses DMA to infiltrate an operating system can be detected using techniques such as those presented by [29] or [28].

Furthermore, memory acquisition can be also used for non-attacking purposes, for example for the transparent acquisition and analysis of volatile memory of a compromised system [8]. Seger et al. [26] presented a memory sampling mechanism based on DMA using a GPU coprocessor as an extension. Schwarz et al. [25] presented an architecture that prevents virtual guest machines accessing memory regions of other virtual guest machines using DMA only by using software and standard hardware. Balogh et al [4] proposed a memory acquisition system which uses DMA based on a custom network interface protocol driver and a network card that can directly send the memory over the network. Chen et al. [9] developed a FPGA based data protection system called sAES which uses DMA to improve throughput and latency of a data protection system.

In this work, we use physical acquisition of specific regions in the volatile memory of a system in order to built an isolated firewall which transparently matches the system's running applications to the network traffic and is able to detect connections from malicious or unknown applications.

2.2 Tamper-resistant Security Architectures

In order to avoid being manipulated by malware or intruders who successfully execute an attack and gain access, some advanced security architectures use tamper-resistant components. These architectures are somehow isolated from the system which they actually protect and monitor, and they also run transparently and unknown to the target.

Many tamper-resistant security architectures use hardware expansions like a special trusted processor booting the micro kernel of the system [30] or particular co-processors only used for monitoring [20] or cryptography [1]. For example, Yashiro et al. [32] propose to use a tamper-resistant chip which does the sensitive operations in a access control scenario on a file system. The most popular architectures are based on a Trusted Platform Module (TPM) which is a dedicated secure crypto-processor issued by the Trusted Computing Group[3] and used for encryption, attestation and sealing of data on a target system. However, tamper-resistant architectures are also used in embedded devices [21].

In recent years, tamper-resistant security architectures which are based on a hypervisor setup are in the center of interest. These architectures are software-based and use a Virtual Machine Monitor (VMM) as a mechanism to guarantee trustworthy isolation of components. Wang et al. [31] propose a combination of both, a hardware-assisted monitor to verify the integrity of a hypervisor and its isolation mechanisms.

Often, these architectures use Virtual Machine Introspection (VMI) in order to analyze the memory of a user virtual machine from another isolated and privileged virtual machine running co-residently on the same hardware [19]. Baiardi et al. [3] proposed a tamper-resistant intrusion detection architecture that merges target monitoring via VMI and network monitoring. Payne et al. [18]

[3] `http://www.trustedcomputinggroup.org/`

proposed an architecture that allows popular security tools to do active monitoring while being isolated in a trusted virtual machine. Srivastava et al. [27] built an isolated firewall which correlates network packets to applications using VMI in a hypervisor setup.

In this work, we secure a target system using an external hardware box or expansion card that regulates the network traffic and additionally uses transparent memory introspection via DMA to match network packets to running applications.

2.3 Memory Introspection

Analysis of memory contents has been explored before in different contexts. Most recently, Virtual Machine Introspection (VMI) was developed as a new technique which uses virtualization and the privileged hypervisor software to analyze memory of guest virtual machine (VM). The ability to analyze memory has been leveraged to detect kernel rootkits [10] or detect malware inside the guest VMs [6]. Such techniques are software-only and do not combine firewall like networking protections with the memory introspection capabilities.

Among hardware-oriented proposals, new architectures have been proposed which leverage extra hardware to perform the memory introspection. Multi-core processors have been extended so that the measurement code can run in a specialized processor local memory [12], and thus not be affected by the malware on the target. Other architectures, e.g. [13, 16], have proposed extra hardware to directly monitor memory bus traffic. Outside of the main processor, [20] has proposed a co-processor based solution where a co-processor performs the monitoring. Also, a special piece of hardware that is connected to one of the DRAM sockets has been proposed to be used to transparently analyze the memory contents as the reads and writes happen [14].

Unfortunately, such hardware additions or modifications are not available today, unlike PCI expansion cards such as FireWire or Thunderbolt available today and used in our project. Moreover, most of the previous projects focus on kernel integrity measurement, whereas SystemWall monitors networking applications.

3 Architecture

The SystemWall is a physical device which is connected to the network in-between the target which it protects and the Internet. Additionally, the SystemWall is connected to an interface of the target computer that allows DMA to the system memory. Figure 1 illustrates the proposed architecture in which the SystemWall is an external standalone device. However, it can be also an internal device added into the target computer like for example a custom PCIe expansion card having two Ethernet interfaces plugged into the target system. When the SystemWall is deployed, it inspects the incoming and outgoing traffic of the target system and matches the traffic to a program running on the system. Currently, the SystemWall focuses on the TCP network protocol.

3.1 Threat Model

We assume a strong threat model where we want protect a computer system even when its applications, OS or hypervisor software may have been compromised. For example, SystemWall should work even when malware has successfully infected the target system and tries to communicate with a malicious remote party. The potential infection can be based on a drive-by exploit targeting an unpatched browser vulnerability or the malware could be a botnet-client trying to connect to the botnet. It could even be a malicious piece of monitoring software performing a phishing attack and trying to send out sensitive information about a user's banking account.

We assume that the hardware-based memory introspection mechanism, such as through a dedicated PCIe expansion card, cannot be compromised as it is separate from the target system and no software on the target system can manipulate it. It is correctly manufactured and hardware itself is not malicious. The firewall software runs on the dedicated external box (which is connected to the hardware-based memory introspection card in the target), or could be part of a custom expansion card in the target. Since the firewall software runs separate from the target platform it cannot be compromised. We assume the firewall software is correctly written.

If system wall is implemented (as in our prototype) on a separate computer, Trusted Platform Module could be used to attest the SystemWall software on startup. If SystemWall is implemented as a custom PCIe card, the firmware will likely be small enough for formal verification, which would given even more confidence in correctness of the SystemWall system.

In our threat model, we do not address physical attacks which means we do not assume an attacker gaining physical access to the target system and being able to remove the SystemWall, or modify the system's hardware parameters, such as the amount of installed memory (DRAM). SystemWall is not able to protect against denial-of-service attacks.

3.2 Boot-up and SystemWall Initialization

SystemWall is independent of the target computer, although may share the same power supply if it is implemented as a expansion card on the motherboard. When the target system is offline, the SystemWall does not have to operate, and thus can be off as well. When the target system boots up, there is no special SystemWall operation. We only assume that at boot up time, SystemWall knows the amount of physically installed memory (DRAM), and that the amount will not change after runtime. The SystemWall can block network connections until the memory acquisition is working, thus any potential malware will not be able to use the network before SystemWall has chance to access memory and look for the malware.

3.3 SystemWall Run-Time

Once the SystemWall can detect the initiation of a new outgoing network connection (SYN sent), it starts the following procedures which are illustrated in Figure 2.

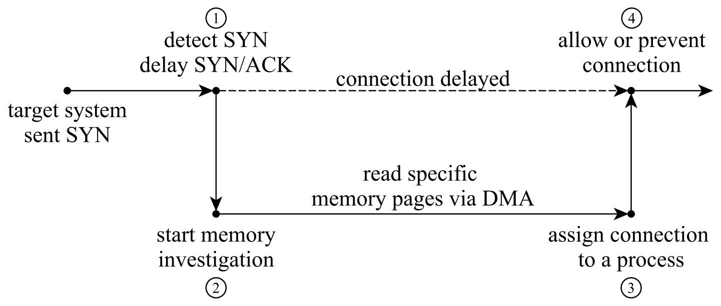

Fig. 2. Once a new outgoing TCP connection initiation is detected (SYN packet), the SystemWall delays the returning SYN/ACK packet while investigating the memory to find the process associated with the connection on the target system.

First, it detects new network connection and delays the SYN/ACK packet sent back from a remote system on the Internet. This way it delays the TCP three-way-handshake completion and the establishment of a new connection.

Second, during the time when the reply is being delayed, the SystemWall starts to transparently read specific memory regions on the target system by exploiting DMA. The target memory regions depend on the installed operating system and a corresponding template which is usually used for forensic investigations on a memory dump. By reading this data, the SystemWall retrieves information about the currently running processes and their currently allocated physical memory regions on the target system. With the help of this information, it matches the new initiated connection to the corresponding running process by matching the source port of the SYN network packet to the port used by the process[4].

Once the SystemWall is able to match the new connection to a process, further checks can be initiated and the integrity of the process can be verified. Finally, the SystemWall decides if the establishment of the new connection is allowed because it is made by a process that is allowed to communicate or if the connection establishment should be prevented because it was initiated by an unknown process.

[4] If the OS is compromised and, for example, there are duplicate data structures that map network ports to applications, SystemWall can scan whole memory of the target system to look for such duplicates and issue a warning.

For some applications, such as web servers, the response delay (while SystemWall checks connection) may have impact on usability. For example, a study [17] found that website users can tolerate around 2s delays in waiting for information retrieval. An alternative to design presented in Figure 2 would be to throttle the responses, while the checking is performed. This way remote parties start receiving information, albeit at a slower pace, while the checking is performed. This, however, is less secure as the unknown process is allowed to send back some information, and in the remainder of the paper we focus on the design which delays the whole connection until all checking is done.

As SystemWall is a separate system and has its own network interface, SystemWall is able to use the network connection to notify the administrator or the target system of any security events. Such interface should be limited, so it cannot be exploited by attackers to compromise SystemWall. In a simplest form, SystemWall could send e-mail notifications to a fixed (administrator) e-mail address without exposing complicated web interface that could be attacked.

SystemWall may also need an interface for updating the firewall software or trusted program white lists. Secure software update is a broad research area with many challenges [5], and secure update of SystemWall is outside of scope of this paper.

3.4 Memory Acquisition

The key part of SystemWall is the memory acquisition which is done through DMA and independent of any software running on the target system. The DMA operations go directly from the SystemWall device (such as a FireWire card), through I/O MMU (if present, such as if the system has Intel's VT-d extensions or AMD's IOMMU technology), to DRAM memory. When target system's OS or hypervisor is non-malicious, then the DMA can proceed easily and verify the integrity system.

If, however, the OS or a hypervisor manipulates the memory contents or the I/O MMU configuration, the SystemWall's access to some of the memory pages can be denied, redirected or false memory content can be presented. SystemWall deals with memory acquisition issues in number of ways and can always block network connections to outside world. If any anomalies are encountered, first action is to temporarily delay the network connections until the issue is resolved. SystemWall can also measure the executable code of the OS or a hypervisor in order to detect any malicious modifications or code. This could be tricked, however, through a number of memory manipulation attacks which we discuss in the following sections. To counter any possible memory manipulation attacks by malicious OS or hypervisor, SystemWall can use a number of detection techniques, for example time-based measurements to detect if an OS or hypervisor is doing something malicious with memory mappings.

Memory Swapping Attacks The software of the target system or a hypervisor may swap memory at some address to disk or another secondary storage and

	16kbyte	**32kbyte**	**64kbyte**	**128kbyte**
memory	0.003 ± 0.001s	0.006 ± 0.001s	0.007 ± 0.001s	0.007 ± 0.001s
hard drive	0.015 ± 0.000s	0.027 ± 0.001s	0.036 ± 0.001s	0.038 ± 0.001s

Table 1. Measured time while reading data from the volatile memory or swapped out data from the hard drive.

replace it with other memory. For example, when SystemWall is trying to scan the system's memory to determine application binary, the OS or hypervisor may temporarily swap the actual (malicious) running application's memory to swap disk and replace it with some (benign) application's memory. Thus, when SystemWall uses DMA to the memory, it will read the benign application's memory and not detect any problem. This, however, requires precise timing on the part of the malicious OS or hypervisor. Nevertheless, it may be possible. If a malicious OS or hypervisor tries to swap memory, it has to execute three steps:

1. Delay the access request of the SystemWall to a memory region through manipulation of the I/O MMU remapping tables, e.g., deny access to the memory region.
2. Swap current memory contents for (benign) memory content from disk (or another storage device).
3. Allow the SystemWall to proceed and to access the target memory region by re-allowing memory access to the memory region.

SystemWall can detect this memory swapping attack by observing memory access errors and access time delay. If access is denied and re-gained after a period of time, SystemWall can detect the time period when memory was not accessible and issue warning that something malicious is going on. Depending on the design of the underlying hardware of the system, a denied DMA access request can result in returning 0s, which can also be detected as an anomaly by the SystemWall. Furthermore, if access is delayed, SystemWall can simply detect the longer access times and this way reveal the memory swapping attack.

In several test-runs we measured time while the system reads small chunks of data from the volatile memory or swapped out data from the hard drive. Clear and constant differences in the measured time can be seen (Table 1) which can be reliably used for anomaly detection.

A more complicated memory swapping attack could be performed using memory in the GPU or other device that is connected to higher speed bus, such as PCI Express, rather than to disk. Nevertheless, even with PCI Express 2.0, the maximum bandwidth is 8GB/s (16 lanes)[5] whereas main memory can have bandwidth over 16GB/s (DDR3)[6]. Including other overheads, memory accesses are at least over 2x faster than going to devices and time differences can be used for anomaly detection.

[5] `https://en.wikipedia.org/wiki/PCI_Express`, accessed Aug. 8, 2014.

[6] `https://en.wikipedia.org/wiki/DDR3_SDRAM`, accessed Aug. 8, 2014.

In-Memory Redirection Attacks Malicious software or a hypervisor could also swap two memory regions in the system's memory (e.g. swap address A1 with A2) or redirect DMA requests of the SystemWall from A1 to other memory pages by remapping the adresses in the I/O MMU. Then, SystemWall would think it is accessing machine address A1, whereas it would actually access A2. In Figure 3, the SystemWall tries to read the physical address 0x04, but is redirected by malicious software to read 0x15. This way, the malicious software can hide data under the address 0x04. The SystemWall cannot detect the in-memory redirection attack based on deviations in access time, like in the previous section. However, from the perspective of the SystemWall, there are now two regions in the memory which store exactly the same data, namely 0x04 and 0x15. This means the SystemWall can search for duplicates.

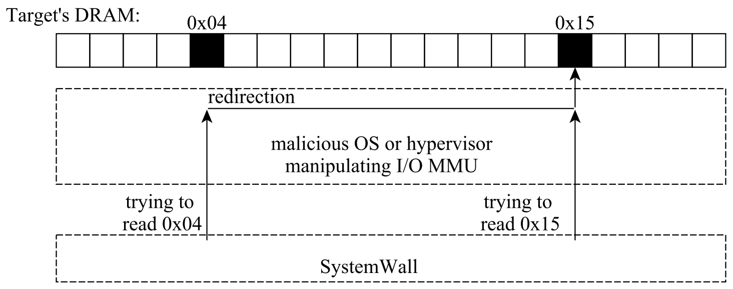

Fig. 3. Example of remapping of requests of the SystemWall to other memory regions by malicious software. The SystemWall reads the same data at 0x04 and 0x15, and can detect duplicates.

In this context, we assume malicious software modified the executable code of a process and stores the original content somewhere else in memory. If the SystemWall wants to read the process's executable code, it is redirected to the original content. In order to detect this attack, the SystemWall uses integrity measurements on the executable code. We assume a SHA1 hash over the original executable code is known. As a potential detection mechanism, the SystemWall executes the following procedures:

- It computes a hash over the process's executable code by reading the target physical addresses via DMA.
- If the hash is valid, the SystemWall subsequently scans the whole memory in chunks of the same size each time computing a hash over the data as well.
- If the same hash occurs in another memory region, a remapping attack was executed which leads to immediate blocking of the process's communication.

At regular time intervals, the SystemWall can execute these procedures and try to detect a potential redirection attack. Scanning for duplicates on a system with 4GB of memory can be performed in our setup in 67.2 ± 2.8 seconds for 512KB of data. This kind of attack detection procedure requires some time, however, it can be halted and continued once the SystemWall needs to perform other operations. The SystemWall continuously tries to detect memory redirection attacks in the background. If other operations need to be performed, the memory redirection attack detection procedure can be halted.

A special case of duplicate code may involve code that is replicated in multiple programs, e.g. statically linked libraries. We assume that in such cases, SystemWall will have sufficient information about the target binaries to be able to count the expected number of code repetitions and detect when there are more than the expected number of code copies when scanning the memory.

Memory Blocking Attacks Usually, some memory regions are "reserved" and used by devices, thus not accessible. A malicious OS or hypervisor could modify the memory map and pretend that there is some device at address $[addr_1, addr_2]$ range. It could then hide some memory contents there so when SystemWall accessed address A1 it would find some benign code or data, while the actual running code or data would be inside $[addr_1, addr_2]$ range.

However, since SystemWall knows the total amount of memory installed, it can analyze memory to try to find any memory region it cannot access. Next, it can use that information and compare with the correct memory configuration of the target system. If the correct or expected memory configuration differs from the one detected, e.g., there is one too many reserved memory regions, then SystemWall can calculate that some malicious action has been taken to create this fake reserved memory region (which could hide some code).

4 Evaluation

In this section, we evaluate a prototype implementation of the SystemWall and discuss the results.

4.1 Prototype Implementation

In an evaluation, we used a system with a Intel Core 2 Duo 2.33 GHz and 3 Gb of memory as the target system for protection. The SystemWall was a second computer system, running Ubuntu Linux 12.04 (64-bit). Both computers were connected via FireWire PCIe cards (LSI FireStorm FW643e2) [15]. Other interfaces like for example Thunderbolt or PCMCIA could be used as well. The components of the SystemWall are programmed in Python and use forensic1394[7] library and the Volatility framework[8] for forensic investigations. The SystemWall also uses Linux iptables with a netfilter script to dynamically delay (and block if needed) network packets.

[7] `https://github.com/wertarbyte/forensic1394`
[8] `https://code.google.com/p/volatility/`

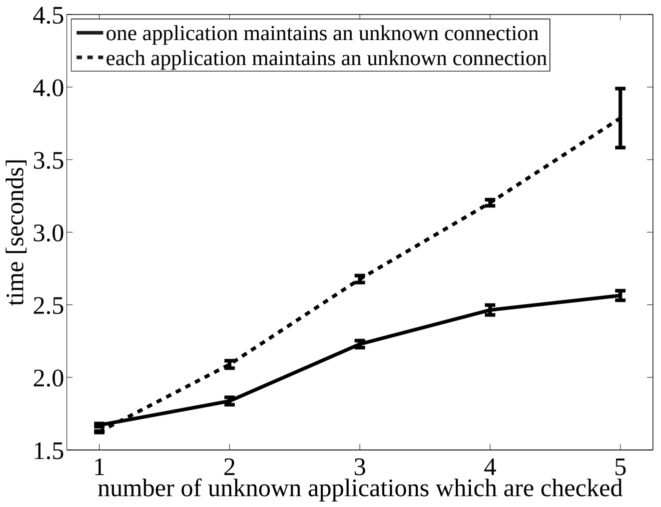

Fig. 4. Time required to decide if a new initiated connection can be allowed or should be blocked based on timely memory investigations (DMA connection overhead, if using FireWire card for example, is not included).

4.2 New Connection Detection and Decision

We investigated the times required for the SystemWall to make a decision if the new initiated connection can be allowed or should be blocked. This is actually also the time for how long the establishment of every new outgoing network connection needs to be delayed. The time required to make a decision should be small, since in this case the usability plays a major role. To accelerate the procedures, the SystemWall can maintain a white-list of known processes belonging to the target's OS's standard installation (e.g. Ubuntu Linux in our case) and which are allowed to communicate. The white-list can be deployed from the beginning, or it can be created stepwise by the SystemWall through an enrollment phase, such as on-line evaluation of the programs and building a white list as the target runs. Figure 4 shows the time of different test-runs (10 test-runs in each configuration) and the corresponding standard deviation.

In the first test-runs (continuous line), a new connection was initiated and the SystemWall matched it to 1 of 5 unknown running processes on the target by investigating the memory via DMA. In the seconds test-runs (dashed line), a new connection was initiated and each of the 5 unknown processes already maintains another connection which needs to be matched as well. In the performed test-runs, the maximum required time was around 4 seconds which is feasibly in our scenario and provides an acceptable level of usability.

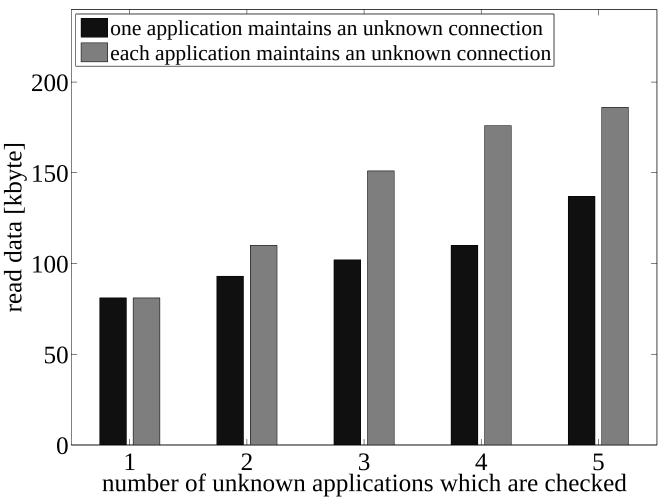

Fig. 5. Amount of memory read by the SystemWall through DMA in order to match a new initiated connection to a running process.

4.3 Memory Throughput Requirements

In further test-runs, we investigated how much data needs to be read through DMA in order decide if a new connection is allowed (Figure 5). The amount of data that needs to be read strongly influences the performance. For this, we used the same configurations as in the previous test-runs (standard Ubuntu installation and up to five unknown additional processes). Once a single new connection was initiated and needed to be matched to 1 up to 5 unknown processes, up to 140 kilobyte was read (black bars). Once a single new connection was initiated and each of the unknown processes maintained a connection, only up to 190 kilobyte was read (grey bars). This is little data and therefore the throughout of the DMA bus cannot be a bottleneck for the SystemWall.

In general, DMA is executed on the PCI bus and therefore could be influenced by other work-loads on the bus running in parallel. In order to investigate the stability of the technique, we executed benchmarks on the target which focused on CPU, memory, file I/O on the storage device and network utilization. In parallel, we executed the procedures of the SystemWall. In each case, the time remain unaffected and stable, since only small amount of data needs to be read. Only if the SystemWall dumped the whole memory of the target system during the benchmarks, deviations in the ranges of $\pm 1\%$ could be seen.

4.4 Process Identification

In order to verify running processes and to create and maintain a white-list of known processes which are allowed to establish connections, investigating infor-

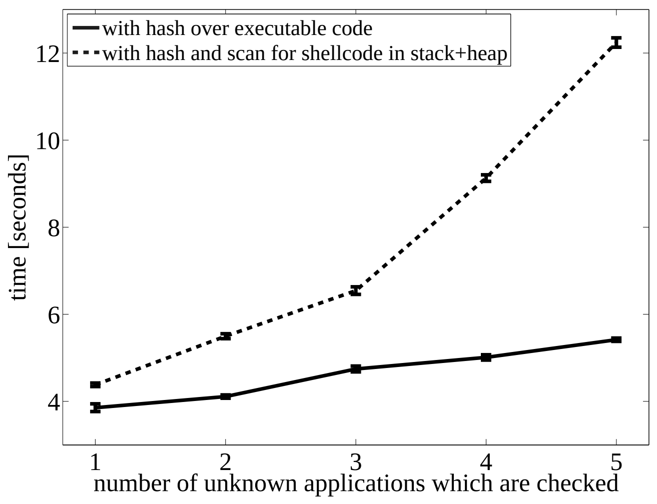

Fig. 6. Time required to decide if a new initiated connection can be allowed or should be blocked using hashes over the executables, and hashes plus scanning for shellcode.

mation of a process like the name or other simple properties is not sufficient. Malware could hide itself by presenting information and properties of known processes. In the next test-runs, the SystemWall computed hashes (SHA1) over the processes' executable codes (ELF) stored in memory in order to verify integrity of the code and to detect potential modifications (Figure 6). Computing the hashes over up to 5 different processes could be checked in less than 5 seconds (continuous line). Furthermore, Figure 6 shows the times for computing the hash and additionally scanning the allocated heap and stack memory of each process for a shellcode signature (long sequences of NOPs) indicating an exploited software vulnerability (dashed line). Shellcode which could be found in these memory regions can be the result of a successfully executed attack against the process that could change its work-flow to do unwanted tasks for example revealing sensitive information. Even if the SystemWall executes these additional in-depth checks, the time increase only up to around 6 seconds for 3 processes and go up to 12 seconds for 5 processes (dashed line). However, the time is strongly depend on the processes and their sizes.

5 Limitations

We believe that hardware assisted memory introspection had a very good potential. Currently, a number of limitations exist. We believe these to be very interesting research opportunities which we and other researchers can tackle and make hardware-assisted memory introspection the new standard in memory introspection work. Physical attacks are not in the scope of our work. However, future design could be created which attempt to deal with physical attacks. An attacker could physically disable or disconnect SystemWall from the system being monitored. Integration of SystemWall into vPro [11] or similar solution could tightly integrate it with the target system's hardware. Integration of SystemWall into the networking infrastructure could also be another research direction. The traffic management could be done at switch or router level. If this "distributed" SystemWall does not receive information form a target (such as when there has been a physical attack), then all traffic can be stopped deeper in the network before the malware or sensitive information has ability to spread.

6 Conclusion

In this work we have presented SystemWall which uses hardware memory introspection to transparently analyze the applications running on a target computer. SystemWall is independent of any software running on the target computer, thus cannot be affected by any malware on the target system, even by a compromised hypervisor. The SystemWall can monitor the applications and control network connections of these applications to prevent spread of malware, viruses or potentially leakage of sensitive files. We have built and evaluated a prototype based on a FireWire card and external SystemWall box. The prototype is able to control network connections of the target system and decide if unknown application's connection can be let through or should be blocked. Our ongoing and future work involves addressing the performance and other challenges presented in the paper.

6.1 SystemWall Code

SystemWall code related to this publication and setup instructions are available online at `http://caslab.eng.yale.edu/code`.

Acknowledgements

We would like to thank anonymous reviewers for their feedback, which helped to improve the final version of this paper. We would like to also thank LSI Corporation for the donation of the two LSI FireStorm FireWire PCIe cards.

References

1. R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov. Cryptographic Processors – A Survey. *Proceedings of the IEEE*, 94:357–369, 2006.
2. D. Aumaitre and C. Devine. Subverting Windows 7 x64 Kernel with DMA attacks. *HITB Security Conference Presentation*, 2010. `http://esec-lab.sogeti.com/dotclear/public/publications/10-hitbamsterdam-dmaattacks.pdf`, accessed Aug. 8, 2014.

3. F. Baiardi and D. Sgandurra. Building Trustworthy Intrusion Detection through VM Introspection. In *Proceedings of the International Symposium on Information Assurance and Security*, pages 209–214, August 2007.
4. S. Balogh and M. Mydlo. New possibilities for memory acquisition by enabling DMA using network card. In *Proceedings of the International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*, pages 635–639, September 2013.
5. A. Bellissimo, J. Burgess, and K. Fu. Secure Software Updates: Disappointments and New Challenges. In *Proceedings of USENIX Hot Topics in Security (HotSec)*, pages 37–43, July 2006.
6. C. Benninger, S. Neville, Y. Yazir, C. Matthews, and Y. Coady. Maitland: Lighter-Weight VM Introspection to Support Cyber-security in the Cloud. In *Proceedings of the International Conference on Cloud Computing (CLOUD)*, pages 471–478, June 2012.
7. R. Breuk and A. Spruyt. Integrating DMA attacks in exploitation frameworks. *Technical Report, System and Network Engineering Research Group, University of Amsterdam*, February 2012.
8. B. D. Carrier and J. Grand. A hardware-based memory acquisition procedure for digital investigations. *Digital Investigation*, 1(1):50–60, 2004.
9. Y. Chen, Y. Wang, Y. Ha, M. Felipe, S. Ren, and K. M. M. Aung. sAES: A high throughput and low latency secure cloud storage with pipelined DMA based PCIe interface. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, pages 374–377, December 2013.
10. T. Fraser, M. Evenson, and W. Arbaugh. VICI – Virtual Machine Introspection for Cognitive Immunity. In *Proceedings of the Annual Computer Security Applications Conference*, pages 87–96, December 2008.
11. Intel vPro Technology. `http://www.intel.com/content/www/us/en/architecture-and-technology/vpro/vpro-technology-general.html`, accessed Aug. 8, 2014.
12. Y. Kinebuchi, S. Butt, V. Ganapathy, L. Iftode, and T. Nakajima. Monitoring Integrity Using Limited Local Memory. *IEEE Transactions on Information Forensics and Security*, pages 1230–1242, July 2013.
13. H. Lee, H. Moon, D. Jang, K. Kim, J. Lee, Y. Paek, and B. B. Kang. KI-Mon: A Hardware-assisted Event-triggered Monitoring Platform for Mutable Kernel Object. In *Proceedings of the USENIX Security Symposium*, pages 511–526, August 2013.
14. Z. Liu, J. Lee, J. Zeng, Y. Wen, Z. Lin, and W. Shi. CPU Transparent Protection of OS Kernel and Hypervisor Integrity with Programmable DRAM. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 392–403, June 2013.
15. FireStorm FW643/FW533 Evaluation Platform. `http://www.lsi.com/downloads/Public/1394\%20Products/1394\%20Products\%20Common\%20Files/LSI-FireStorm-PB.pdf`, accessed Aug. 8, 2014.
16. H. Moon, H. Lee, J. Lee, K. Kim, Y. Paek, and B. B. Kang. Vigilare: Toward Snoop-based Kernel Integrity Monitor. In *Proceedings of the Conference on Computer and Communications Security*, pages 28–37, October 2012.
17. F. F.-H. Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.
18. B. Payne, M. Carbone, M. Sharif, and W. Lee. Lares: An Architecture for Secure Active Monitoring Using Virtualization. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 233–247, May 2008.

19. B. Payne, M. de Carbone, and W. Lee. Secure and Flexible Monitoring of Virtual Machines. In *Proceedings of the Annual Computer Security Applications Conference*, pages 385–397, December 2007.
20. N. L. Petroni, Jr., T. Fraser, J. Molina, and W. A. Arbaugh. Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor. In *Proceedings of the USENIX Security Symposium*, pages 13–13, August 2004.
21. S. Ravi, A. Raghunathan, and S. Chakradhar. Tamper resistance mechanisms for secure embedded systems. In *Proceedings of the International Conference on VLSI Design*, pages 605–611, January 2004.
22. J. Rutkowska. Beyond The CPU: Defeating Hardware Based RAM Acquisition. *Black Hat DC Presentation*, 2007. `http://www.blackhat.com/presentations/bh-dc-07/Rutkowska/Presentation/bh-dc-07-Rutkowska-up.pdf`, accessed Aug. 8, 2014.
23. F. Sang, E. Lacombe, V. Nicomette, and Y. Deswarte. Exploiting an I/OMMU vulnerability. In *Proceedings of the International Conference on Malicious and Unwanted Software (MALWARE)*, pages 7–14, October 2010.
24. F. Sang, V. Nicomette, and Y. Deswarte. I/O Attacks in Intel PC-based Architectures and Countermeasures. In *Proceedings of the SysSec Workshop (SysSec)*, pages 19–26, July 2011.
25. O. Schwarz and C. Gehrmann. Securing DMA through virtualization. In *Proceedings of the Workshop on Complexity in Engineering (COMPENG)*, pages 1–6, June 2012.
26. M. Seeger and S. Wolthusen. Towards Concurrent Data Sampling Using GPU Coprocessing. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*, pages 557–563, August 2012.
27. A. Srivastava and J. Giffin. Tamper-Resistant, Application-Aware Blocking of Malicious Network Connections. In *Recent Advances in Intrusion Detection*, volume 5230 of *Lecture Notes in Computer Science*, pages 39–58. Springer, 2008.
28. P. Stewin and I. Bystrov. Understanding DMA Malware. In *Proceedings of the Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 7591, pages 21–41, July 2013.
29. P. Stewin, J.-P. Seifert, and C. Mulliner. Poster: Towards detecting DMA malware. In *Conference on Computer and Communications Security*, pages 857–860, October 2011.
30. G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. AEGIS: Architecture for Tamper-evident and Tamper-resistant Processing. In *Proceedings of the International Conference on Supercomputing*, pages 160–171, June 2003.
31. J. Wang, A. Stavrou, and A. Ghosh. HyperCheck: A Hardware-assisted Integrity Monitor. In *Proceedings of International Conference on Recent Advances in Intrusion Detection*, pages 158–177, September 2010.
32. T. Yashiro, M. Bessho, S. Kobayashi, N. Koshizuka, and K. Sakamura. T-Kernel/SS: A Secure Filesystem with Access Control Protection Using Tamper-Resistant Chip. In *Computer Software and Applications Conference Workshops*, pages 134–139, July 2010.
33. F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz. Fast and Memory-efficient Regular Expression Matching for Deep Packet Inspection. In *Proceedings of the Symposium on Architecture for Networking and Communications Systems*, pages 93–102, December 2006.