Air Force Institute of Technology
AFIT Scholar

Theses and Dissertations | Student Graduate Works

3-22-2012

Magnesium Object Manager Sandbox, A More Effective Sandbox Method for Windows 7

Martin A. Gilligan

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the OS and Networks Commons

Recommended Citation

Gilligan, Martin A., "Magnesium Object Manager Sandbox, A More Effective Sandbox Method for Windows 7" (2012). *Theses and Dissertations*. 1111.
https://scholar.afit.edu/etd/1111

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.

MAGNESIUM OBJECT MANAGER SANDBOX, A MORE EFFECTIVE SANDBOX METHOD FOR WINDOWS 7

THESIS

Martin A. Gilligan, Second Lieutenant, USAF

AFIT/GCE/ENG/12-05

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
22 MAR 2012		

4. TITLE AND SUBTITLE	
Magnesium Object Manager Sandbox, A More Effective Sandbox Method for Windows 7	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)	
Martin Gilligan	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Air Force Institute of Technology,Graduate School of Engineering and Management (AFIT/EN),2950 Hobson Way,Wright-Patterson AFB,OH,45433-7765	**AFIT/GCE/ENG/12-05**

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

The original document contains color images.

14. ABSTRACT

A major issue in computer security is limiting the affects a program can have on a computer. One way is to place the program into a sandbox, a limited environment. Many attempts have been made to create a sandbox that maintains the usability of a program and effectively limits the effects of the program. Sandboxes that limit the resources programs can access, have succeeded. To test the effectiveness of a sandbox that limits the resources a program can access on Windows 7, the Magnesium Object Manager Sandbox (MOMS) is created. MOMS uses a kernel mode Windows component to monitor and limit the access rights to every resource. Based on the performance data of a set of test programs, running with and without MOMS, and with different hardware configurations, the hardware configuration and MOMS has an impact to performance a normal user probably will not notice. For the exploits run against two of the test programs, none of the associated payloads successfully ran. While these tests are promising, they are limited in scope and further testing is required to increase their scope. Furthermore, based on analysis of MOMS, vulnerabilities exist, but they are straightforward to fix with further development.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**104**	

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States

AFIT/GCE/ENG/12-05

MAGNESIUM OBJECT MANAGER SANDBOX, A MORE EFFECTIVE SANDBOX METHOD FOR WINDOWS 7

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Insitute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Martin A. Gilligan, B.S.C.E., B.S.E.E.

Second Lieutenant, USAF

March 2012

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT/GCE/ENG/12-05

MAGNESIUM OBJECT MANAGER SANDBOX, A MORE EFFECTIVE SANDBOX METHOD FOR WINDOWS 7

Martin A. Gilligan, B.S.C.E., B.S.E.E.
Second Lieutenant, USAF

Approved:

// signed //

Lt Col Jeffrey W. Humphries, PhD (Chairman)	Date

// signed //

Barry E. Mullins, PhD (Member)	Date

// signed //

William B. Kimball (Member)	Date

AFIT/GCE/ENG/12-05

Abstract

A major issue in computer security is limiting the affects a program can have on a computer. One way of accomplishing this is to place the program into a limited environment called a sandbox. Many attempts have been made to create an effective sandbox, one that effectively limits the affects a program can have, yet does not make the program unusable. The sandboxes based around intercepting system calls have historically not been effective; however, sandboxes that limit the resources programs can access, have been effective. To test the effectiveness of a sandbox that limits the resources a program can access on a Windows 7 computer, a sandbox, Magnesium Object Manager Sandbox (MOMS), that uses the Object Manager (OM) callback functionality is created. The OM is the kernel mode Windows component that facilitates access to every resource; third party drivers can monitor and limit the access rights to those resources by registering a function to be called by the OM whenever a program first accesses a resource.

Performance data is collected on a set of test programs, running with and without MOMS, and with different hardware configurations. Based on this data, MOMS has a negligible impact, an impact a normal user probably will not notice, to the performance of the test programs, and the hardware configuration also has a negligible impact on performance, with or without MOMS. To test the effectiveness of MOMS, exploits are run against a subset of the test programs and whether the associated payload was successful is recorded. None of the payloads successfully ran, which indicates MOMS can be an effective sandbox. While these tests of the efficiency and effectiveness of MOMS are promising, they are limited in scope and further testing is required in order to increase their scope. Furthermore, MOMS is analyzed to identify possible vulnerabilities it may have. While there are some, they are all straightforward to fix with further development.

AFIT/GCE/ENG/12-05

I would like to thank my friends and family for keeping me sane and supporting me throughout this experience.
I would also like to thank my advisor and committee members for their invaluable assistance in completing my first thesis.

Table of Contents

Page

Abstract iv

Dedication v

List of Figures viii

List of Tables x

List of Acronyms xi

1 Introduction 1

2 Related Work 5
  2.1 Windows 7 32-bit in Detail 7
    2.1.1 Security Reference Monitor 7
    2.1.2 Object Manager 10
      2.1.2.1 Object Manager Directory 12
      2.1.2.2 Object Header 14
      2.1.2.3 Object Types 17
      2.1.2.4 Object Manager Callbacks 29

3 Methodology 33
  3.1 Goals and Approaches 34
    3.1.1 Efficient Operation 34
    3.1.2 Accurate Object Allow List 34
    3.1.3 Ensure the Enforcement Mode Cannot be Circumvented 35
  3.2 System Under Test 35
    3.2.1 Associated Process List Generation 35
    3.2.2 Rules Generation Mode 36
    3.2.3 Enforcement Mode 37
  3.3 System Services 38
  3.4 Workload 38
  3.5 Performance Metrics 39
  3.6 System Parameters 39
  3.7 System Factors and Levels 40
  3.8 Evaluation Techniques 41
  3.9 Experimental Design 44
  3.10 Summary 44

4 Results and Discussion 45
4.1 Internal Relationships of the Data 46
4.2 Determine Data Distribution 50
4.3 Effect of Factors on Performance 51
4.4 Effect of Magnesium Object Manager Sandbox on Peformance 52
4.5 Effectiveness of Magnesium Object Manager Sandbox 54

5 Conclusion and Future Work 59
5.1 Alternative Sandbox Methods 60
5.1.1 Automatic Program Confinement to Private Namespaces 60

Appendix A: Autocorrelation Plots 62

Appendix B: Distribution Plots 68

Appendix C: Effect of Hardware Configuration on Performance 74

Appendix D: Performance Impact of Magnesium Object Manager Sandbox 80

Bibliography 88

List of Figures

Figure		Page
2.1	Windows Architecture	8
2.2	Access Mask Structure	9
2.3	Handle Structure	11
2.4	Object Manager Directory Structure	13
2.5	The Object Header Structure	15
2.6	Key Object Related Structures Required for Key Name Lookup	21
2.7	Example of the Relationship of the Key Object Related Structures	22
2.8	Functions and Data Structures of the Object Manager Callbacks	31
3.1	Magnesium Object Manager Sandbox	36
4.1	Lag Plots for 7-Zip	47
4.2	Lag Plots for Excel, Spreadsheet 1	47
4.3	Lag Plots for PowerPoint, Slide Show 2	48
4.4	Lag Plots for Word	49
A.1	Autocorrelation Plots for 7-Zip	62
A.2	Autocorrelation Plots for Acrobat Reader	63
A.3	Autocorrelation Plots for Excel	64
A.4	Autocorrelation Plots for Internet Explorer	65
A.5	Autocorrelation Plots for PowerPoint	66
A.6	Autocorrelation Plots for Word	67
B.1	Distribution Plot For 7-Zip	68
B.2	Distribution Plot For Acrobat Reader	69
B.3	Distribution Plot For Excel	70
B.4	Distribution Plot For Internet Explorer	71
B.5	Distribution Plot For PowerPoint	72

B.6 Distribution Plot For Word 73
C.1 Hardware Configuration Performance Results for 7-Zip 74
C.2 Hardware Configuration Performance Results for Acrobat Reader 75
C.3 Hardware Configuration Performance Results for Excel 76
C.4 Hardware Configuration Performance Results for Internet Explorer 77
C.5 Hardware Configuration Performance Results for PowerPoint 78
C.6 Hardware Configuration Performance Results for Word 79
D.1 Performance Impact of MOMS: 1 Core, 2 GB 80
D.2 Performance Impact of MOMS: 1 Core, 3 GB 82
D.3 Performance Impact of MOMS: 2 Cores, 2 GB 84
D.4 Performance Impact of MOMS: 2 Cores, 3 GB 86

List of Tables

Table Page

2.1 Generic Rights 9

2.2 Standard Rights 10

2.3 Directory Object Specific Rights 18

2.4 Synchronization Object Specific Rights 19

2.5 File Object Specific Rights 20

2.6 Key Object Specific Rights 23

2.7 Process Object Specific Rights 24

2.8 Section Object Specific Rights 25

2.9 Thread Object Specific Rights 26

2.10 Token Object Specific Rights 28

2.11 WindowStation Object Specific Rights 29

2.12 Desktop Object Specific Rights 30

3.1 System Factors and Levels for MOMS 42

4.1 Summary of the Impact the Rules Generation Mode (RGM) has on Program Performance 55

4.2 Successfullnes of Exploits with the Enforcement Mode (EnfM) Running 56

D.1 Performance Impact Results of MOMS: 1 Core, 2 GB 81

D.2 Performance Impact Results of MOMS: 1 Core, 3 GB 83

D.3 Performance Impact Results of MOMS: 2 Core, 2 GB 85

D.4 Performance Impact Results of MOMS: 2 Core, 3 GB 87

List of Acronyms

ACE Access Control Entry 7

APL Associated Process List 33

CM Configuration Manager 21

DACL Discretionary Access Control List 7

EnfM Enforcement Mode 33

HD hard drive 40

MOMS Magnesium Object Manager Sandbox 33

OAL Object Allow List 33

OM Object Manager 7

ORL Operation Record List 33

OS operating system 5

PID Process Identifier 35

PWL Program Watch List 33

RGM Rules Generation Mode 33

SELinux *Security-Enhanced Linux* 6

SID security identifier 7

SSD solid state drive 40

SRM Security Reference Monitor 7

MAGNESIUM OBJECT MANAGER SANDBOX, A MORE EFFECTIVE SANDBOX METHOD FOR WINDOWS 7

1 Introduction

People use many programs on their computers in order to accomplish specific tasks. The programs are expected to complete those tasks, and only those tasks, but how can that expectation be enforced? One way is to place the program into a sandboxed, or limited, environment, thereby restricting the tasks the program can perform. Previous sandbox implementations have been made, but some information about how operating systems (OSs) work is needed before they can be explained.

Modern, mainstream, OSs have two modes of operation, kernel mode and user mode. Kernel mode contains the core of the OS and provides functionality, such as access to the file system, to the processes in the user mode. User mode provides a restricted environment for "normal" processes (processes that do not modify or extend the OS itself) to run in. In user mode, processes cannot directly access other processes, unless the other process specifically allows it, and they cannot directly access the kernel mode; they interact with the kernel mode by making system calls. When a user mode process makes a system call, it requests the OS to perform an action, such as writing to or reading from a file, on its behalf.

The previous implementations have relied on intercepting system calls and then allowing or denying the system call based on the sequence of previous system calls or based on the parameters associated with the system call. The problem with allowing or denying a system call based on previous ones is that it often leads to too many false positives for the sandbox to be useful. Sandboxes that make the determination of whether

to allow or deny a system call based on the associated parameters often are too ineffective because of the number of different values for the parameters. Other sandbox implementations restrict the resources a process can access based on other mechanisms, such as using built-in components of the OS. *Security-Enhanced Linux* (*SELinux*) and *FreeBSD Jails* are examples of this latter method implemented on Linux and Unix, but non have been implemented on Windows, to the knowledge of the author. To explore the viability of this latter method on Windows, a proof-of-concept implementation, Magnesium Object Manager Sandbox (MOMS), is created that uses the Object Manager (OM) to restrict the resources sandboxed processes can access.

Windows 7 32-bit, and other Windows OSs, internally, represent every resource as an object, which is a kernel mode structure that contains the necessary information to represent the underlying resource. The OM manages all the objects and facilitates every access to the objects. To interact with an object, a user mode process must have a handle to it. A handle is a reference to an object. When a user mode process wants to access an object, it requests a handle to it from the OM, along with the access rights it wants over it. If the permissions the process possesses allow the requested access rights over the object, the OM issues the process a handle to the object and stores the access rights the process requested. When the process wants to perform an operation on an object it has a handle to, the process requests the OS to perform the action on the object, by sending the OS the handle and the desired action. If the access rights the process initially requested, when it obtained the handle, are sufficient to allow the desired action, the OS performs it.

The access rights associated with a handle can be restricted further, by third party code running in kernel mode, by registering a callback function with the OM. When a callback function is registered with the OM, the OM calls the function every time a process receives a handle to an object. MOMS uses this functionality in both of its modes of operation: the Rules Generation Mode (RGM) and the Enforcement Mode (EnfM). The

RGM logs the objects, and the corresponding access rights, that a program receives over the course of its operation. From this log, a list of objects, and the maximum allowed access rights, is generated by an administrator and enforced by the EnfM.

The goals of MOMS is for it to run efficiently and to be effective. In order to run efficiently, MOMS must not add an excessive amount of overhead, which would cause users not to use it, because it slows down their system too much. MOMS must be effective in two aspects, the RGM must accurately log the objects a program accesses, along with the corresponding access rights, and the EnfM must ensure a program cannot alter the system in such a way that it is able to access an object it should not be able to. Since the logging of object accesses by the RGM and the object access checking by the EnfM both run in linear time, MOMS is expected to run efficiently. Since every object access request is viewable by MOMS, the RGM and the EnfM can log and check, respectively, every object access; therefore, MOMS is expected to be effective,

Once MOMS is implemented, its efficiency and effectiveness is determined. The operation of the RGM is very similar to that of the EnfM, so the RGM will be used to indicate the overall efficiency of MOMS. To determine the efficiency of MOMS, performance metrics will be collected on a set of test programs running in and out of the RGM with various hardware configurations and program input. To determine the effect the amount of available memory and the number of processor cores have on performance, for each combination of program, program input, and whether the program is running with or without the RGM, the performance metrics of the various hardware configurations will be compared. To determine the performance impact of the RGM, the performance metrics with and without the RGM will be compared, for each combination of program, program input, and hardware configuration. Finally, the effectiveness of MOMS will be determined by running a set of exploits against a subset of the programs and recording whether the EnfM prevents the exploit from executing its payload. Furthermore, possible

vulnerabilities in MOMS will be identified and analyzed to determine the effect they have on the security that MOMS offers.

2 Related Work

"A sandbox is an environment in which the actions of a process are restricted according to a security policy" [10, p. 444]. This definition includes methods such as individual programs sandboxing themselves, the operating system (OS) creating a sandbox around programs, and the entire OS running in a sandboxed environment through hardware virtualization. Program level sandboxes allow the program author to just sandbox the most vulnerable portions, but this method leaves the unsandboxed portions of the program vulnerable, are vulnerable to kernel exploits and users overriding restrictions, and require the program to be rewritten. Operating system (OS) level sandboxes do not require programs to be rewritten, but are still vulnerable to kernel exploits and are also potentially vulnerable to users overriding restrictions. Hardware virtualization provides the most secure environment, since the host OS is not vulnerable to exploits of the sandboxed OS, but it does present significant usability issues [19].

To gain a better understanding of the current state of sandboxes, some currently available sandboxes and research sandboxes will be presented. Since the sandbox being developed for this paper is an OS level sandbox, only sandboxes that operate at this level will be presented. Much research has been conducted on sandboxes in the academic world. The most common approach is for the sandbox to monitor system calls and determine their actions based on them. Sandboxes such as [27] and [29] generate a model to represent the sequence of system calls a program makes, and then if the program deviates from the model it prevents the action. Sandboxes such as [16], [28], and [43] extend this approach to not only look at the system calls a program makes, but to look at the effect they produce, such as the actions they take on files and registry keys; similarly, [38] extends this approach to dynamically sandbox programs based on the data, and source of the data, they access. Other research efforts go as far as running device drivers in a virtualized environment to improve system security and reliability [49].

In addition to research sandboxes, many production sandboxes have been created. For the Unix and Linux OSs, and their many variants, there are three popular sandboxes: *Security-Enhanced Linux* (*SELinux*), *AppArmor*, and *FreeBSD Jails*. *SELinux* provides a framework to enforce a security policy, which can limit the resources a process can access based on mandatory access controls, which enforce access controls regardless of who the user is (*SELinux* treats the “root” super-user the same as regular users) [45]. *AppArmor* is an alternative to *SELinux* that also limits the resources, and the permissions to those resources, that programs can access [7]. Another alternative, *FreeBSD Jails* restricts a process to a given directory subtree and assigns each jail its own hostname and static IP address. Furthermore, Jails can have their own set of users; however, these users are restricted to the jailed environment, so even the root user inside of a jail cannot perform operations outside of the jailed environment [40].

On the Windows platform, several companies produce sandboxes: *Comodo Firewall*, *avast! Pro Antivirus*, and *Sandboxie*. *Comodo Firewall* contains a sandbox that automatically sandboxes programs. Comodo maintains a list of known safe programs and known malware. *Comodo Firewall* allows programs on the known safe programs list to run outside the sandbox and removes programs that are on the known malware list [11]. *avast! Pro Antivirus* does not automatically sandbox programs, but allows users to run programs in a virtual (sandboxed) environment and prompts users to do so for suspicious programs [9]. *Sandboxie* takes a different approach than the previous two, it focuses on preventing sandboxed programs from making permanent changes to the system by intercepting changes the sandboxed programs make and redirects them to the sandbox environment. This allows *Sandboxie* to delete any changes the sandboxed program makes to the system [21].

2.1 Windows 7 32-bit in Detail

Windows 7 consists of many components that provide the services and functionality essential for the OS. These components reside in both user and kernel mode, as shown in Figure 2.1. Due to hardware and software protections, user mode code cannot directly access kernel mode memory, while kernel mode code can directly access user mode memory. Furthermore, code running in one process cannot directly access the memory of another process, unless the other process allows it, such as through shared memory. Each user mode component, as well as each user mode program, runs in its own process.

The Object Manager (OM) manages and facilitates access to all the objects on the system, which represent system resources. The Security Reference Monitor (SRM) provides the infrastructure to secure the objects. All other components rely on these two components to interact with system resources in a secure manner [30].

2.1.1 Security Reference Monitor. Windows 7 32-bit uses a user-based security model, implemented by the SRM, where each user has a set of actions they are allowed to take, such as reading a file or writing a registry key. The SRM uses access tokens, Token objects, to determine the actions a process can take. Access tokens are created when a user logs on and contains the security identifier (SID) of the account for the user and any groups the user belongs to. SIDs uniquely identify users and groups on the system. When an user starts a process, the process receives the access token for the user. Therefore, each process a user starts has the full rights of the user by default, although restricted access tokens can be given to a process [36].

Every securable object has a Discretionary Access Control List (DACL) that lists the access rights the various SIDs can have over the object. The DACL consists of a set of Access Control Entrys (ACEs). Each ACE contains the allowed or denied access rights for a SID. When a process wants to access an object, it indicates the access rights it wants

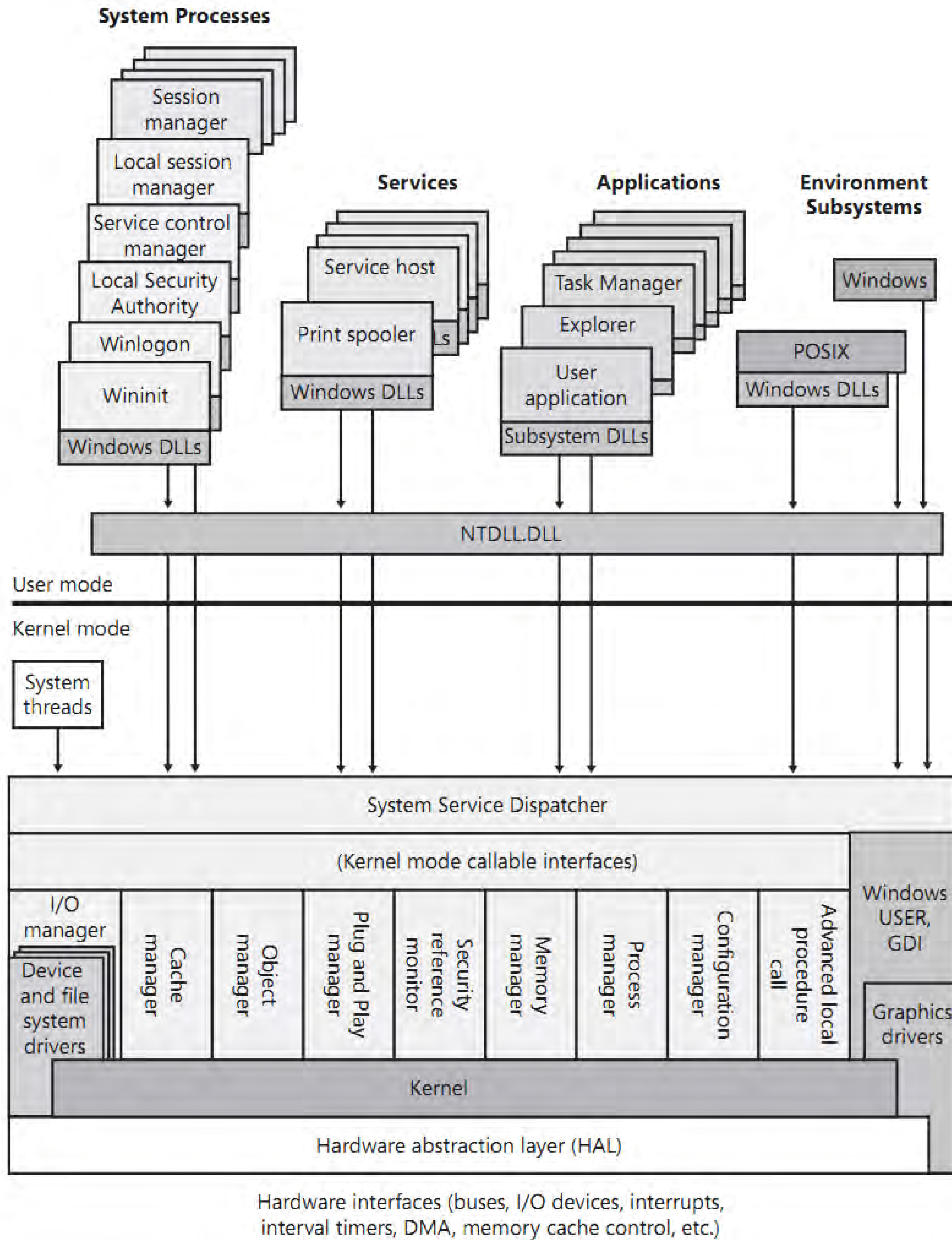


Figure 2.1: Windows Architecture [41, Figure 2-3]

over the object with an access mask. In order to determine whether the access rights should be allowed, the SRM compares the SIDs that apply to the process to the DACL of the object the process wants to access and determines whether or not the access, as indicated by the access mask, should be allowed [36].

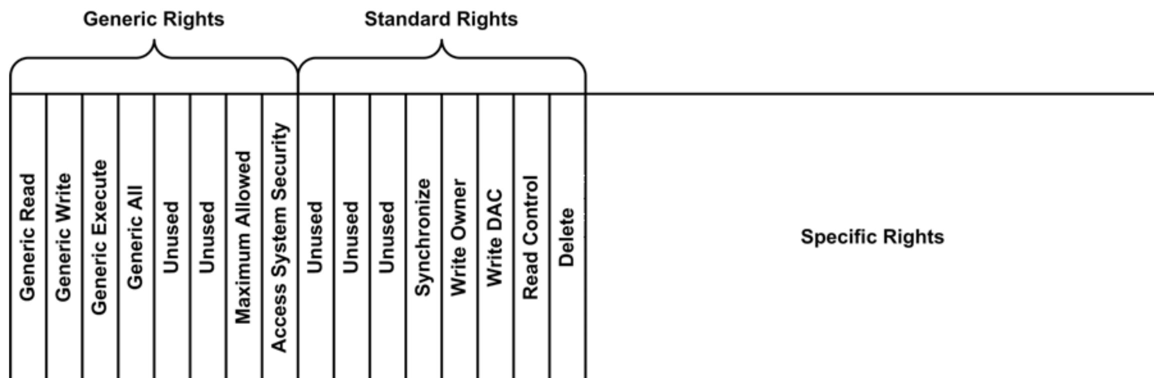


Figure 2.2: Access Mask Structure [5]

Table 2.1: Generic Rights [5]

Generic Right Name	Generic Right Mapping
GENERIC_READ	Rights to read the object
GENERIC_WRITE	Rights to write the object
GENERIC_EXECUTE	Rights to execute, or alternatively view, the object
GENERIC_ALL	Rights to read, write, and execute the object
MAXIMUM_ALLOWED [51]	Grants the maximum allowed access rights when making an access check
ACCESS_SYSTEM_SECURITY [42]	Set or read the SACL in the security descriptor of an object

Table 2.2: Standard Rights [5]

Standard Right Name	Allowed Action
SYNCHRONIZE	Wait on the object
WRITE_OWNER	Modify the owner SID of the object
WRITE_DAC	Modify the security information for the object
READ_CONTROL	Read the security information for the object
DELETE	Delete the object

Access masks, a 32-bit field, indicate the desired access rights a process wants over an object. Access masks have three fields, as depicted in Figure 2.2: generic, standard, and specific. Each generic access right maps to a set of standard and specific rights; the mapping depends on the object type. Table 2.1 lists the meaning of each generic right field. Table 2.2 describes the standard Rights, rights common to all object types. The specific rights depend on the object type and are described below, for the object types that are pertinent to this paper.

2.1.2 Object Manager [41, p. 133-170]. The OM in the Windows 7 OS, as well as others, provides a common and centralized method for the kernel to manage access to all system resources. The OM represents system resources as objects. In this instance, an object is a data structure that contains pertinent information about a resource, not an object in the object-oriented programming sense. The OM retains the object structures, in memory, until no process requires access to them and provides a way to access objects by name. Each object consists of a header and a body. The header contains information common to all objects. There are many object types that represent various types of resources (files, keys, synchronization primitives, etc.). The object body is specific to, and the same for, each object type; it contains information common to each object of the

corresponding object type. The OM uses the header to manage all objects in a uniform manner. Each object type is implemented by an executive component, which uses the object body to manage the objects of the type it implements.

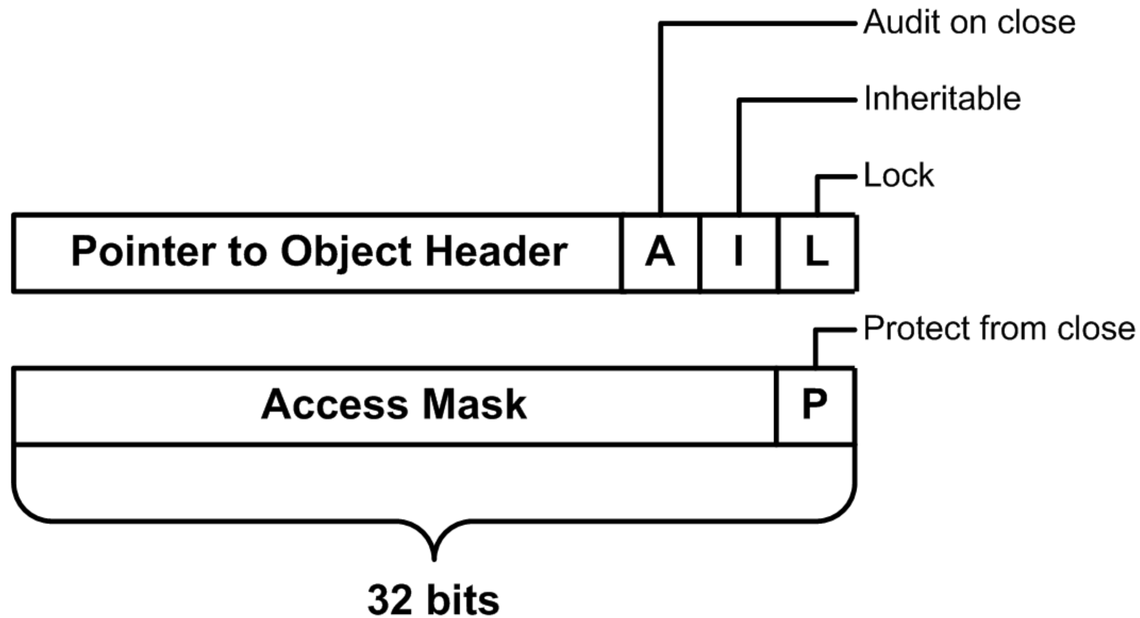


Figure 2.3: Handle Structure [41, Figure 3-19]

The object data structures reside in kernel space and are accessed in two ways: by pointer or by handle. User-mode processes access objects by handle, while processes in kernel-mode can access objects by pointer or by handle. A kernel-mode process can determine the location of the object structure without the aid of the OM, so it can access any object without access checks; however, when an object is accessed by pointer, through the OM, the OM retains the object until access is no longer required. A handle is an executive structure, depicted in Figure 2.3, that the OM uses to determine the location of the object being accessed and the access rights a process possesses over an object (determined by the *ACCESS_MASK* field). The other fields of the handle structure are not important to this research. When a kernel mode process accesses an object by handle,

the process receives a pointer to the object, the process can decide whether or not the access check should be conducted, and the OM retains the object until it is no longer being accessed. When an user mode process accesses an object by handle, the user mode process passes a handle to the object, and the actions to be performed, to the kernel, then the kernel conducts the actions on the object, if the user mode process has sufficient permissions.

When access to a resource is requested, if there is not already an object that represents it, one is created. The object is kept until all processes indicate that they no longer need access to the object, at which time the object is deleted. A process can receive a handle to an object, and thereby have access to it, in one of three ways: when the process creates the object, by opening an object by name, and by receiving a duplicate handle from a process that already has a handle to the object (occurs when one process inherits handles from another process or when one process explicitly duplicates a handle and gives it to another process). Two processes share an object when both open the same object by name or when one process receives a handle from the other process through duplication.

2.1.2.1 Object Manager Directory. The OM organizes all the named objects into a directory structure, as depicted in Figure 2.4, that is much like a file system, and is referred to as the OM namespace. All named objects, except Key, File, and Process objects, have their names in the OM namespace. Whenever an object is referenced by name, the name of the object is passed to the OM, which then traverses the directory until it either finds the object or determines that the object does not exist in the OM namespace. While Key and File object names are not in the OM namespace, their names are rooted there. When a Key or File object is referenced by name, the name of the Key or File object is passed to the OM, which traverses the OM namespace portion of the object name then passes the rest of the object name to the appropriate component: the Configuration

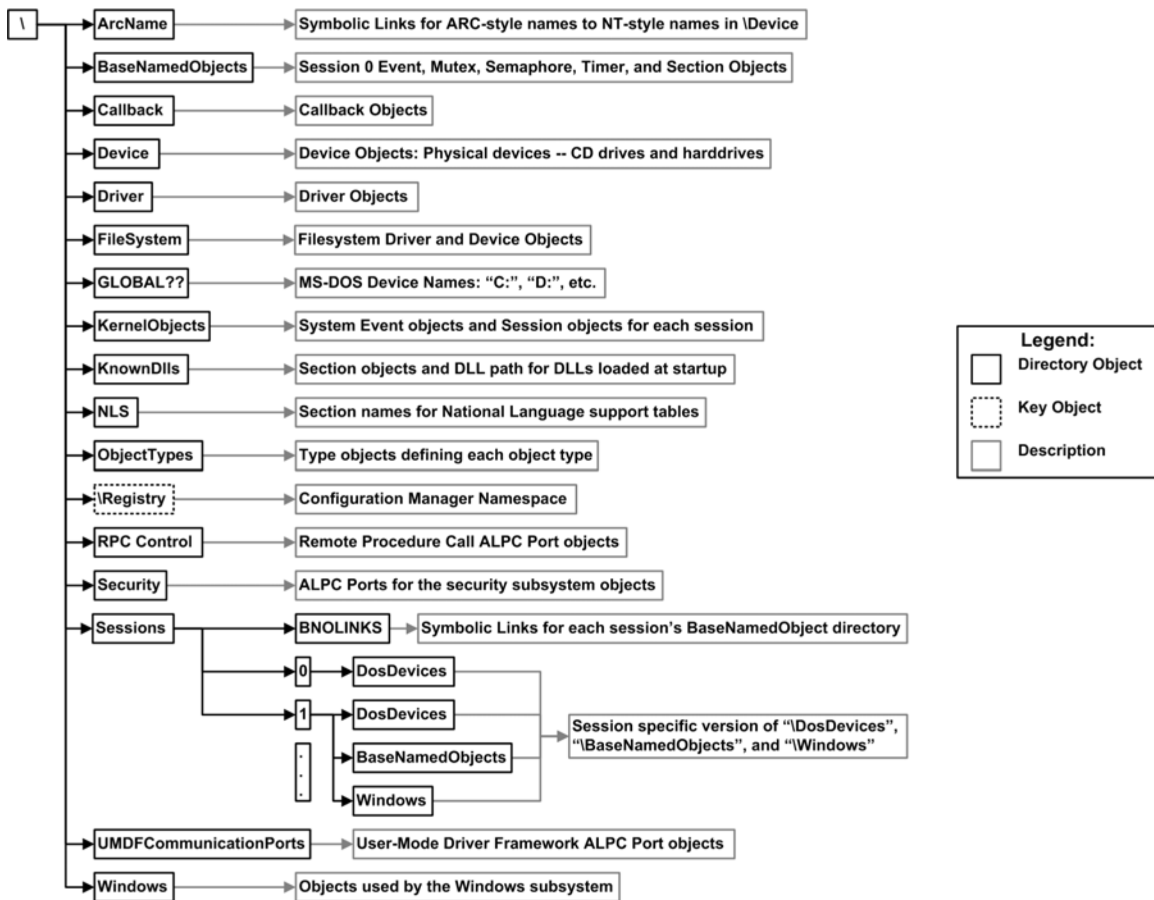


Figure 2.4: Object Manager Directory Structure [41, 33, Table 3-14]

Manager for Key objects and the File Manager for File objects. The Key and File object name lookups will be further detailed below in the object type descriptions.

2.1.2.2 Object Header.

The Object Header provides information common to every object. The OM uses this structure to manage all the objects in a general manner. As shown in Figure 2.5, the object header contains eleven mandatory fields and five optional headers. The object header immediately precedes the object body of the object that it refers to. The optional fields included with a specific object header are indicated by the *InfoMask* field, as described below, and they immediately precede the mandatory fields in the order indicated in Figure 2.5. Below are the descriptions of each field [53, 41, p. 139-140].

PointerCount and HandleCount The OM uses *PointerCount* and *HandleCount* to determine when it is safe to delete an object. When a resource is referenced, and there is no object that represents it, the OM creates one. Each object is retained until both *PointerCount* and *HandleCount* are zero, at which time the OM deletes the object. When the object is referenced or dereferenced by pointer the OM increments or decrements, respectively, *PointerCount*. When the object is referenced or dereferenced by handle the OM increments or decrements, respectively, both *PointerCount* and *HandleCount*. So *PointerCount* is a count of the number of pointer and handle references to the object, while *HandleCount* is only a count of the number of handle references to the object.

NextToFree This field is undocumented.

Lock A per-object lock used to ensure the object structure is not modified by two processes at the same time.

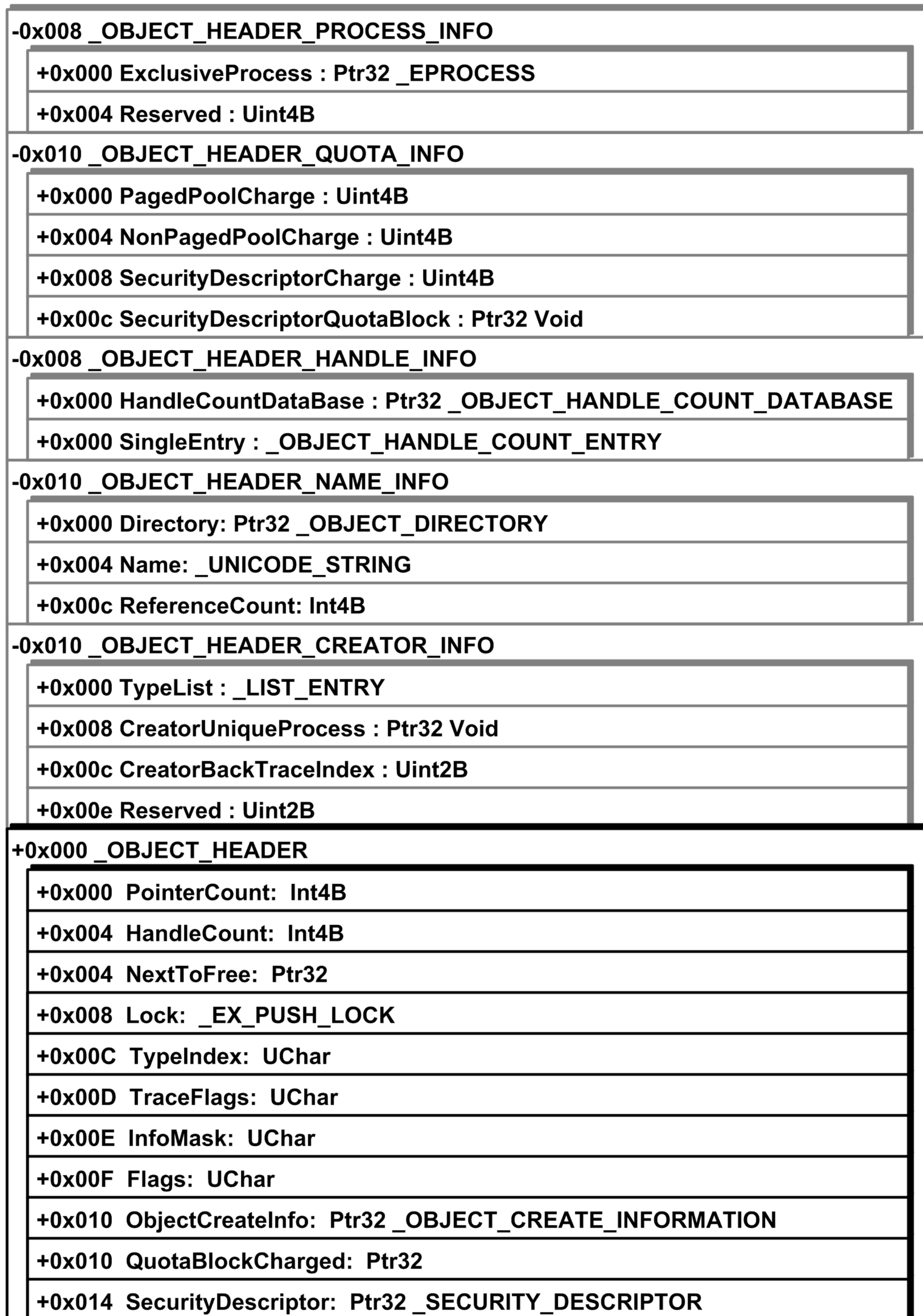


Figure 2.5: The Object Header Structure

TypeIndex An index into a global array of pointers to the object types. The value in this field indicates the object type of the object.

TraceFlags This field contains information related to tracing object references and dereferences during debugging.

InfoMask This field indicates which optional headers, if any, are included with the object header. The presence of each of the five optional headers is indicated by a bit in the *InfoMask* field. The order of the included optional headers, as well as the position of the corresponding bit (lowest to highest), are the same as Figure 2.5 indicates. Below are the descriptions of each optional header:

Creator Information (Bit Pos 0) This optional header links the object to all other objects of the same type and contains a pointer to the process that created the object. This optional header is only included when the Type object for the object has the *MaintainTypeList* flag set.

Name Information (Bit Pos 1) For objects created with a name, this optional header contains the name of the object and a pointer to its place in the object namespace.

Handle Information (Bit Pos 2) Contains a list of the processes that have a handle to the object open. This header is only included when the Type object for the object has the *MaintainHandleCount* flag set.

Quota Information (Bit Pos 3) This optional header contains the resource charges against a process when it opens a handle to the object. This optional header is included when the quota of the object differs from the default quota of the object type and for other special cases.

Process Information (Bit Pos 4) This optional header is active if it is opened to be exclusive to a single process and it contains a pointer to the process.

Flags This is a set of attributes for the object that the OM uses to manage the object internally. The flags are described in.

ObjectCreateInfo Maintains information about the process that created the object and a list that links the object to other objects of the same type. This field is only valid when the Type object for the object has the *MaintainTypeList* flag set.

QuotaBlockCharged Contains the resource charges against a process when it opens a handle to the object. This field is valid when the quota of the object differs from the default quota of the object type and for other special cases.

SecurityDescriptor This is a pointer to the permissions for this object and is only valid for named objects.

2.1.2.3 Object Types. Since Windows represents every resource as an object, there are many object types, because there are many types of resources. Object types are themselves represented by the Type object, an object itself. Windows allows the creation of object types through undocumented functions within the kernel. The Type object contains information common to and aggregate statistics about all objects of that type, information the OM requires in order to manage objects of that type, and default settings for objects of that type. There are 42 object types by default; however, only 20 of them are directly accessible in user mode through API functions [41, p. 136-137]. The object types that are only directly available to the kernel are the following: Adapter, ALPC Port, Callback, Controller, DebugObject, Device, Driver, EtwConsumer, EtwRegistration, EventPair, FilterCommunicationPort, FilterConnectionPort, IoCompletionReserve, KeyedEvent, PcwObject, PowerRequest, Profile, Session, SymbolicLink, Type, UserApcReserve, WmiGuid. They are detailed below. Since this sandbox is intended to only sandbox user-space programs, only the object types that are available in user-space will be detailed.

Directory Directory objects provide the directory structure for the OM in a similar way as a directory in a file system; however, the OM directory structure is different and independent from the file structure. Limiting the directories in the OM that a process has access to can provide additional, although course, security. Table 2.3 lists the specific rights that apply to Directory objects.

Table 2.3: Directory Object Specific Rights [33, 3]

Access Mask	Specific Right Name Specific Right Description
0x0001	DIRECTORY_QUERY Object Directory Query
0x0002	DIRECTORY_TRAVERSE Object Directory Name Lookup
0x0004	DIRECTORY_CREATE_OBJECT Object Directory Create Name
0x0008	DIRECTORY_CREATE_SUBDIRECTORY Object Directory Create Subdirectory
0x000F	DIRECTORY_ALL_ACCESS All of the Object Directory Specific Rights

Synchronization Object (Event, Mutex, Semaphore, Timer) Synchronization objects allow multiple threads to synchronize their execution, such as limiting the number of threads that can concurrently execute a section of code. While synchronization objects do not pose a direct security risk, that is they cannot allow access to

resources a process should not have access to, they can lead to denial of service. A compromise of a synchronization object can lead to a denial of service if the compromise prevents a thread from executing when it should. While these object types are specifically created for synchronization, other object types can facilitate synchronization [2]. The specific rights that pertain to synchronization objects are listed in Table 2.4.

Table 2.4: Synchronization Object Specific Rights [47, 3]

Access Mask	Specific Right Name	Specific Right Description
Event		
0x0001	EVENT_QUERY_STATE	Query Event Object State
0x0002	EVENT_MODIFY_STATE	Modify Event Object State
Mutex		
0x0001	MUTEX_MODIFY_STATE	Modify Mutex Object State
Semaphore		
0x0001	SEMAPHORE_QUERY_STATE	Query Semaphore Object State
0x0002	SEMAPHORE_MODIFY_STATE	Modify Semaphore Object State
Timer		
0x0001	TIMER_QUERY_STATE	Query Timer Object State
0x0002	Timer_MODIFY_STATE	Modify Timer Object State

File The OM represents files, directories, and pipes with the File object. File objects do not have a name in the OM namespace, but they do have an internal name in the File object structure, which is the name of the file, directory, or pipe it represents. The

internal name is anchored to the OM namespace by the name of the drive it resides on, which is located in the *Device* directory. Table 2.5 lists the specific rights for File objects.

Table 2.5: File Object Specific Rights [54, 12, 3]

Access Mask	Type	Specific Right Name	Specific Right Description
0x0001	F, P	FILE_READ_DATA	Read Data from the File / Pipe
	D	FILE_LIST_DIRECTORY	List the Files in the Directory
0x0002	F, P	FILE_WRITE_DATA	Write Data to the File / Pipe
	D	FILE_ADD_FILE	Add Files to the Directory
0x0004	F	FILE_APPEND_DATA	Append Data to the File
	D	FILE_ADD_SUBDIRECTORY	Add Directories to the Directory
	P	FILE_CREATE_PIPE_INSTANCE	Create a Named Pipe Instance
0x0008	F, D	FILE_READ_EA	Read Extended Attributes
0x0010	F, D	FILE_WRITE_EA	Write Extended Attributes
0x0020	F	FILE_EXECUTE	Read File Data Into Memory
	D	FILE_TRAVERSE	Traverse the Directory
0x0040	D	FILE_DELETE_CHILD	Delete a File or Directory from the Directory
0x0080	A	FILE_READ_ATTRIBUTES	Read the Attributes of the File
0x0100	A	FILE_WRITE_ATTRIBUTES	Write the Attributes of the File

F: File **D: Directory** **P: Pipe**

IoCompletion This object provides a mechanism to alert processes when an I/O operation finishes. Therefore, this object does not need additional protections, since the I/O operations are protected [22].

Job Job objects provide a mechanism to manage processes as a group, such as setting security attributes, suspending them, terminating them, etc. Job objects only group processes, which have their own security, so if the processes are secured, the job objects do not need to be secured [25].

Key The Key object represents registry keys and Table 2.6 contains the specific rights for Key objects. Key objects, like File objects do not have a name in the OM namespace, rather they have their own internal name. The OM directory has a Key object attached to the root of the directory named *Registry*. To lookup a registry key by name, the OM receives the name of the key from the Configuration Manager (CM) (the component responsible for the registry) and starts looking up the name. It first encounters the Key named *Registry*, at which time it sends the rest of the name of the key to the CM to finish the lookup.

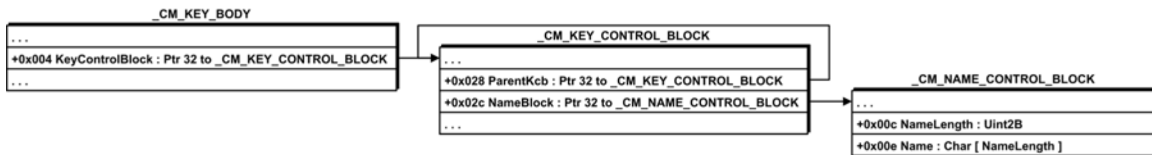


Figure 2.6: Key Object Related Structures Required for Key Name Lookup

As indicated above, there are two levels for registry keys: the OM level and the CM level. The Key object, called `_CM_KEY_BODY` in Figure 2.6, refers to a specific registry key, while the `_CM_KEY_CONTROL_BLOCK` and the `_CM_NAME_CONTROL_BLOCK` refer to a subkey. Each `_CM_KEY_CONTROL_BLOCK` points

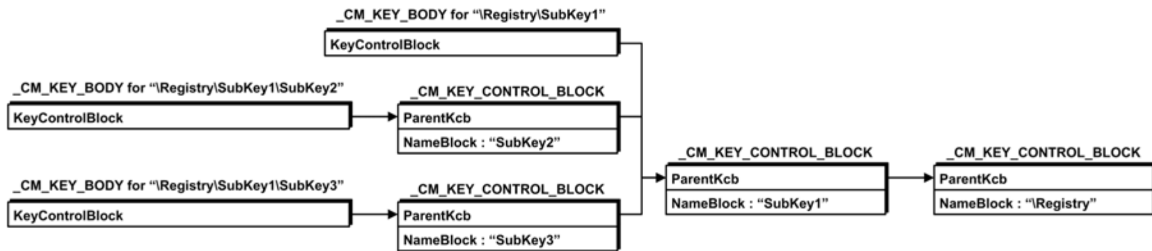


Figure 2.7: Example of the Relationship of the Key Object Related Structures

to the next subkey of its name; keys that have common beginning key names share _CM_KEY_CONTROL_BLOCKs for the shared subkeys. For example, as depicted in Figure 2.7, there is a Key object, _CM_KEY_BODY, for each of the three registry keys: *\Registry\SubKey1*, *\Registry\SubKey1\SubKey2*, and *\Registry\SubKey1\SubKey3*. Each of the three Key objects point to a different _CM_KEY_CONTROL_BLOCK, but have the *SubKey1* and *\Registry* _CM_KEY_CONTROL_BLOCK in common in their chain. Finally, the key name is anchored in the root directory of the OM [41, p. 276-277].

Process The Process object contains the information needed to manage processes. When a process creates another process, it receives a handle to the new process with full access rights. The specific rights for the Process object are in Table 2.7.

Section Section objects contain the necessary information to manage a section of memory. A memory section can map to a file or to the page file. Furthermore, processes can share access to memory sections through views. Views allow a process to access a portion of a memory section and defines what actions a process can take on that portion of memory [44]. Table 2.8 lists the specific rights for Section objects.

Table 2.6: Key Object Specific Rights [39]

Access Mask	Specific Right Name	Specific Right Description
0x0001	KEY_QUERY_VALUE	Query Registry Key Values
0x0002	KEY_SET_VALUE	Create, Delete, or Set Registry Key Values
0x0004	KEY_CREATE_SUB_KEY	Create a Subkey of a Registry Key
0x0008	KEY_ENUMERATE_SUB_KEYS	Enumerate the Subkeys of a Registry Key
0x0010	KEY_NOTIFY	Allows Receipt of Change Notifications for a Registry Key or its SubKeys
0x0020	KEY_CREATE_LINK	Reserved for system use
0x0100	KEY_WOW64_64KEY	Indicates a 64-bit application should operate on the 64-bit registry view - ignored by 32-bit Windows
0x0200	KEY_WOW64_32KEY	Indicates a 64-bit application should operate on the 64-bit registry view - ignored by 32-bit Windows

Table 2.7: Process Object Specific Rights [35, 3]

Access Mask	**Specific Right Name**	**Specific Right Description**
0x0001	PROCESS_TERMINATE	Terminate the process
0x0002	PROCESS_CREATE_THREAD	Create a thread associated with the process
0x0004	PROCESS_SET_SESSIONID	
0x0008	PROCESS_VM_OPERATION	Modify the address space of the process
0x0010	PROCESS_VM_READ	Read from the address space of the process
0x0020	PROCESS_VM_WRITE	Write to the address space of the process
0x0040	PROCESS_DUP_HANDLE	Duplicate handles to or from the process
0x0080	PROCESS_CREATE_PROCESS	Create a child process of the process
0x0100	PROCESS_SET_QUOTA	Set the working set size for the process
0x0200	PROCESS_SET_INFORMATION	Modify process settings
0x0400	PROCESS_QUERY_INFORMATION	Query process settings
0x0800	PROCESS_SUSPEND_RESUME	Suspend or resume the process
0x1000	PROCESS_QUERY_LIMITED_INFORMATION	

Table 2.8: Section Object Specific Rights [55, 3]

Access Mask	Specific Right Name	Specific Right Description
0x0001	SECTION_QUERY	Query the Section object for information about the section
0x0002	SECTION_MAP_WRITE	Write views of the section
0x0004	SECTION_MAP_READ	Read views of the section
0x0008	SECTION_MAP_EXECUTE	Execute views of the section
0x0010	SECTION_EXTEND_SIZE	Dynamically extend the section size
0x0020	SECTION_MAP_EXECUTE_EXPLICIT	Undocumented

Thread The Thread object represents information required for the system to manage the thread. Threads share the handle table of the process they are associated with; however, a thread can gain access to more objects than their associated process through impersonation tokens and a process can create a thread in the context of another process. The ability for a process to make the previous changes, and others, are specified through the specific rights for Thread objects and are detailed in Table 2.9.

Kernel Transaction Objects (Enlistment, Resource Manager, Transaction Manager, Transaction) The Kernel Transaction Manager provides support for executing multiple operations as an atomic transaction. Transactions allow multiple operations to be conducted so that if any of the operations fail, all of the operations are undone. These objects provide the kernel level support for this functionality [41, p. 240-241].

Table 2.9: Thread Object Specific Rights [35, 50, 3]

Access Mask	Specific Right Name	Specific Right Description
0x0001	THREAD_TERMINATE	Terminate the thread
0x0002	THREAD_SUSPEND_RESUME	Suspend or resume the thread
0x0004	THREAD_ALERT	Undocumented
0x0008	THREAD_GET_CONTEXT	Query the execution context of the thread
0x0010	THREAD_SET_CONTEXT	Modify the execution context of the thread
0x0020	THREAD_SET_INFORMATION	Modify the thread settings
0x0040	THREAD_QUERY_INFORMATION	Query the thread settings
0x0080	THREAD_SET_THREAD_TOKEN	Set the impersonation token for a thread
0x0100	THREAD_IMPERSONATE	Directly use the security information of a thread
0x0200	THREAD_DIRECT_IMPERSONATION	Allows a server thread to impersonate a client
0x0400	THREAD_SET_LIMITED_INFORMATION	Modify a limited set of thread settings
0x0800	THREAD_QUERY_LIMITED_INFORMATION	Query a limited set of thread settings

Since transactions are wrappers for other operations, no additional security must be applied to them, since the operations themselves possess their own security.

Token Token objects contain the access control lists, security identifiers, and other items the SRM requires to enforce user-based security. A process can alter the contents of a Token, thereby altering the permissions a process has over an object [36]. Therefore, the actions a process can take on a Token object must be controlled and the specific rights are defined in Table 2.10.

TpWorkerFactory The TpWorkerFactory is an object for kernel level support for thread pools. Thread pools allow a process to create a dynamic number of threads, that are managed by the kernel. Since the TpWorkerFactory is essentially a container for threads, much the same way as a Job object is a container for processes, additional security is not needed [41, p. 386-390].

WindowStation and Desktop Window stations contain a clipboard, an atom table, and a set of Desktop objects. There is only one interactive window station per session called *WinSta0*. An atom table is a 16-bit integer, atom, to string look-up table [1]. Each Window Station has three Desktops by default: the logon desktop, the default desktop, and the screensaver desktop. Each Desktop has a set of windows. Processes on a Desktop can only communicate with other processes on the same Desktop, through the Desktop mechanisms, such as Window messages. Window Stations and Desktops do not allow processes on them to communicate between separate Window Stations and Desktops; however, processes can control the Window Station or Desktop they are on and can affect other processes on the same Window Station or Desktop, so the access rights a process has to a Window Station or Desktop must be controlled [4]. The specific rights for WindowStation objects

Table 2.10: Token Object Specific Rights [6, 3]

Access Mask	Specific Right Name	Specific Right Description
0x0001	TOKEN_ASSIGN_PRIMARY	Attach a primary token to the process
0x0002	TOKEN_DUPLICATE	Duplicate the access token
0x0004	TOKEN_IMPERSONATE	Attach an impersonation token to the process
0x0008	TOKEN_QUERY	Query the access token
0x0010	TOKEN_QUERY_SOURCE	Query the source of the access token
0x0020	TOKEN_ADJUST_PRIVILEGES	Enable or disable the privileges in the access token
0x0040	TOKEN_ADJUST_GROUPS	Adjust the attributes of the groups in the access token
0x0080	TOKEN_ADJUST_DEFAULT	Change the default owner, primary group, or DACL of the access token
0x0100	TOKEN_ADJUST_SESSIONID	Adjust the session ID of the access token

are defined in Table 2.11 and the specific rights for Desktop objects are defined in Table 2.12.

Table 2.11: WindowStation Object Specific Rights [3]

Access Mask	Specific Right Name	Specific Right Description
0x0001	WINSTA_ENUMDESKTOPS	Enumerate existing Desktop objects
0x0002	WINSTA_READATTRIBUTES	Read the attributes of the WindowStation object
0x0004	WINSTA_ACCESSCLIPBOARD	Use the clipboard
0x0008	WINSTA_CREATEDESKTOP	Create a Desktop object on the Window Station
0x0010	WINSTA_WRITEATTRIBUTES	Modify the attributes of the WindowStation object
0x0020	WINSTA_ACCESSGLOBALATOMS	Modify global atoms
0x0040	WINSTA_EXITWINDOWS	Close the Window Station or shutdown the system
0x0100	WINSTA_ENUMERATE	Enumerate the Window Station
0x0200	WINSTA_READSCREEN	Access screen contents

2.1.2.4 Object Manager Callbacks. Starting in Windows Vista, and continuing with Windows 7, Microsoft added callback function capabilities to the OM. The callbacks allow a function to be called before or after an object is created or duplicated. The standard functionality only allows callback functions to be registered for

Table 2.12: Desktop Object Specific Rights

Access Mask	Specific Right Name	Specific Right Description
0x0001	DESKTOP_READOBJECTS	Read objects on the desktop
0x0002	DESKTOP_CREATEWINDOW	Create a window on the desktop
0x0004	DESKTOP_CREATEMENU	Create a menu on the desktop
0x0008	DESKTOP_HOOKCONTROL	Establish one of the window hooks
0x0010	DESKTOP_JOURNALRECORD	Perform journal recording on the desktop
0x0020	DESKTOP_JOURNALPLAYBACK	Perform journal playback on the desktop
0x0040	DESKTOP_ENUMERATE	Enumerate the desktop
0x0080	DESKTOP_WRITEOBJECTS	Write objects on the desktop
0x0100	DESKTOP_SWITCHDESKTOP	Activate the desktop

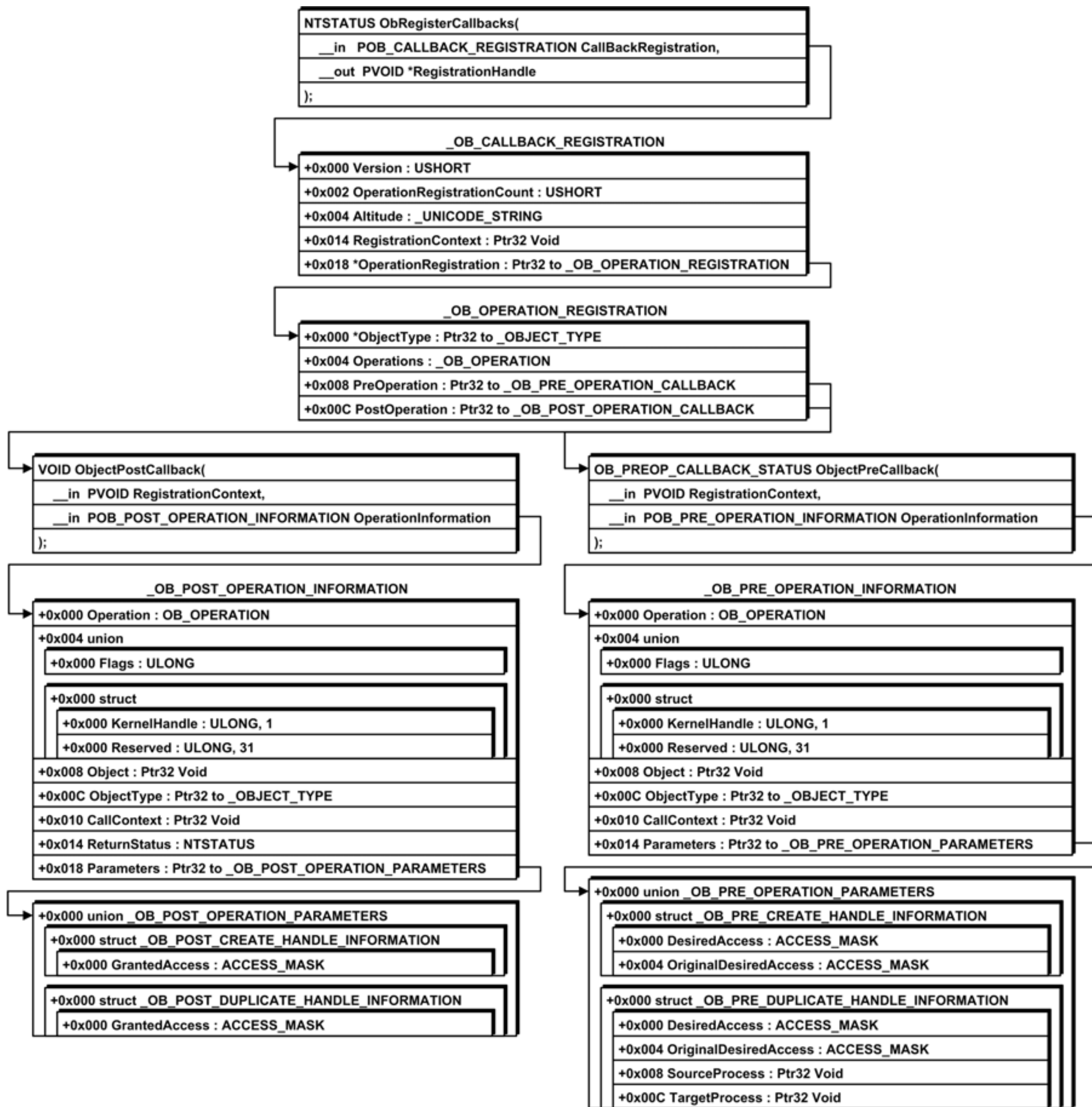


Figure 2.8: Functions and Data Structures of the Object Manager Callbacks

Process and Thread objects; however, any object type can be enabled to have callbacks registered for it by setting the *SupportsObjectCallbacks* bit in the OBJECT_TYPE structure for that object to one. The OM callbacks allow the restriction of the access rights a process receives over an object, but does not allow the object access to be directly blocked (it can be indirectly blocked by removing all access rights).

To use the OM callback functionality, one must register a pre or post callback function with the ObRegisterCallbacks function. The ObRegisterCallbacks function has an _OB_CALLBACK_REGISTRATION structure as input, as shown in Figure 2.8, which contains a pointer to an array of _OB_OPERATION_REGISTRATION structures. Each _OB_OPERATION_REGISTRATION structure contains fields that indicate when the callbacks are triggered: the object type the callback function(s) are called for, whether the function(s) are called before or after the operation, and whether the function(s) should be called when a handle to the object is created or duplicated. In addition, the structure contains the function that should be called for each object type before and after the operation (if a function should not be called, the corresponding field is set to **NULL**).

The main difference between the callback functions lies between the ObjectPreCallback and ObjectPostCallback functions and the create and duplicate modes of the ObjectPreCallback function. The ObjectPreCallback function contain two ACCESS_MASKs, one for the original access the process desires for the object (*OriginalDesiredAccess*) and the access the filter allows the process to the object (*DesiredAccess*; the *DesiredAccess* must have a subset of the access rights in *OriginalDesiredAccess*). The ObjectPostCallback function contains one ACCESS_MASK for the actual access granted to the process for the object. Within the ObjectPreCallback function, the _OB_PRE_DUPLICATE_HANDLE_INFORMATION structure contains a pointer to the process that the handle is being copied from and the process that it is being copied to; the _OB_PRE_CREATE_HANDLE_INFORMATION structure does not.

3 Methodology

Magnesium Object Manager Sandbox (MOMS) monitors and restricts objects that programs, such as Internet Explorer, access. The term program, in this case, refers to all the processes that are required for what a normal user conceptually views as a program to run. MOMS contains a list of all the programs, the Program Watch List (PWL), it should sandbox. Each program in the PWL has a main process, the process used to start the program; for example, “iexplore.exe” is the main process for Internet Explorer. For each program in the PWL, MOMS maintains a list of the processes, Associated Process List (APL), associated with the program.

MOMS has two modes of operation: the Rules Generation Mode (RGM) and the Enforcement Mode (EnfM). The RGM determines the resources each program wants access to and runs on an administration computer, a computer with Windows 7 32-bit free of viruses or any other code unintended to be executed. The RGM produces a list of the objects, the Operation Record List (ORL), and corresponding access rights, a program accesses by monitoring handle creation and duplication with the OM callback functionality. The EnfM runs on production computers with Windows 7 32-bit installed. It uses the same OM callback functionality that the RGM uses to monitor handle creation and duplication requests; instead of logging each request, for each process in the APL, it limits the access rights the process receives over the object to those in the Object Allow List (OAL), which is based on the ORL.

Of the 42 object types present in Windows 7 32-bit, thirteen of them are monitored by MOMS: Directory, Event, Mutex, Semaphore, Timer, File, Key, Process, Section, Thread, Token, WindowStation, and Desktop. The other object types are not monitored because they are either not available to user mode, since the API does not make them available [41, Table 3-5], or they are used to manage other object types that are monitored, such as Job objects.

This section discusses the goals of the individual components of MOMS and the approaches to evaluate those goals.

3.1 Goals and Approaches

The goal of this research is to determine the effectiveness and efficiency of MOMS. The RGM sacrifices some speed for efficient object access storage by updating the `ACCESS_MASK` for objects in the access list, in linear time based on the number of objects in the ORL; therefore, the RGM should run efficiently. The EnfM maximizes both effectiveness and efficiency. The EnfM intercepts each request for access to resources and allows or restricts the request based on the OAL. The request interception runs in constant time and the access check runs in linear time based on the number of objects in the OAL of the program. Since the RGM and EnfM run in linear time and checks each access request, MOMS is expected to be efficient as a whole and effective at preventing payloads associated with exploits from running.

3.1.1 Efficient Operation. To be practical and accepted, the RGM and the EnfM must not add an excessive overhead, a noticeable slowdown, to the operation of the system. Since the operation of the RGM and the EnfM are very similar, the RGM logs access requests and the EnfM allows or restricts access requests, the performance of the EnfM will be very similar to that of the RGM. Therefore, the efficiency of MOMS will be determined by comparing the performance, the execution time or speed, of a set of programs in and out of the RGM, with varying hardware.

3.1.2 Accurate Object Allow List. Another goal for the RGM is that it generates an accurate OAL. This is essential because the security of the system depends on this and less manual configuration is required. To test this goal, a set of programs are ran in and out of the EnfM with the OAL. The accuracy is determined by the number of exploits whose payloads successfully execute without the EnfM versus with the EnfM.

3.1.3 Ensure the Enforcement Mode Cannot be Circumvented.

The last goal ensures the EnfM cannot be circumvented via side channels. For MOMS, a side channel exists when a process can access an object it should not be able to, or in a way it should not be able to, by altering the system in a way that makes MOMS determine the access should be allowed. This goal cannot be fully evaluated with current software because although MOMS could prevent software from using side channels that were written without considering this sandbox, there could be an easily exploitable flaw in MOMS that would render the sandbox ineffective. Therefore, MOMS is evaluated analytically identifying potential vulnerabilities in MOMS and then determining the impact they may have, as well as any possible mitigations for them.

3.2 System Under Test

The system under test (Figure 3.1), MOMS, consists of two Components Under Test (CUT), the RGM and the EnfM, as well as two supporting components, the administration computer and the production computer.

3.2.1 Associated Process List Generation.

The APL contains the Process Identifiers (PIDs) that are associated with the main process. A PID is a numerical unique identifier for processes. Also, each Process object contains the PID of the process that created it, the parent of the process. Whenever a handle to a Process object is created or duplicated, the PID of that process is added to the APL of a watched process if the name of the process is the same as the name of the main process or if the PID of the parent of the process is in the APL. Processes are added to the APL in this fashion, because this method associates any process that a main process starts, directly or indirectly, with the main process.

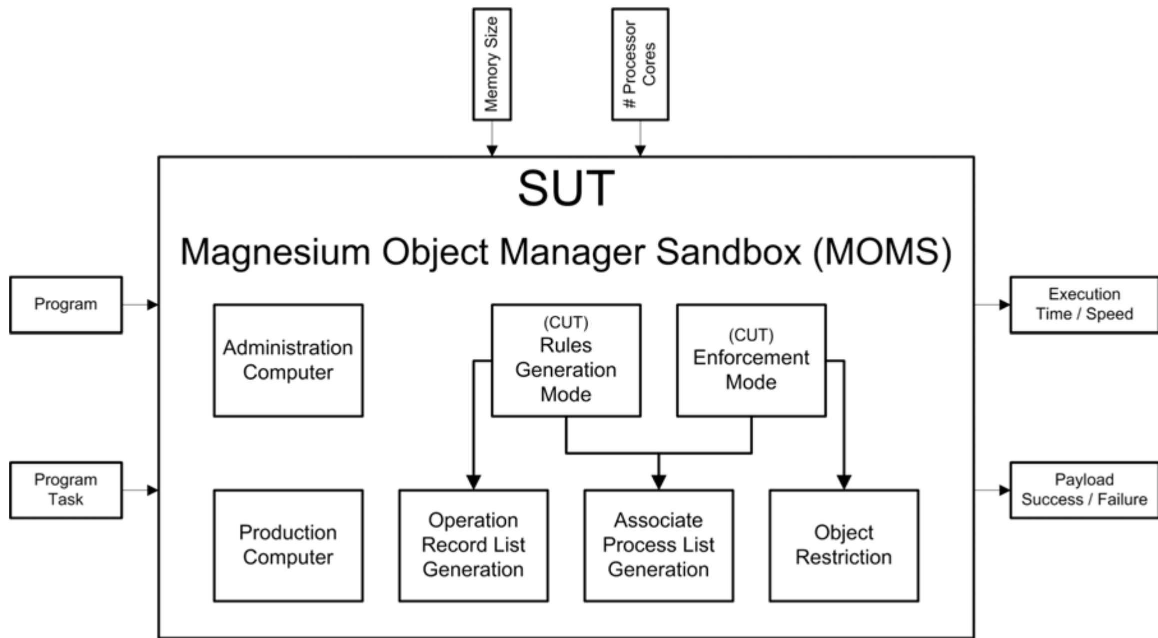


Figure 3.1: Magnesium Object Manager Sandbox

3.2.2 Rules Generation Mode. In the RGM, an ORL is maintained for each program in the PWL. For create operations, if the process creating the handle (the current process) is in the APL of a PWL, the operation is added to the ORL of that program. For duplicate operations, if the process the handle is being duplicated to is in the APL of a PWL, and it is not being duplicated from a process in the same APL, the operation is added to the ORL of that program. If the handle is being duplicated from a process in the APL, the operation is not added to the ORL of that program because the program would already have access to the object in order to be able to duplicate a handle to the object. If the object already exists in the list, the `ACCESS_MASK` of the object is OR'ed with the `ACCESS_MASK` of the object already in the list. Unnamed objects are not included in the ORL. This produces a list of the objects the program accesses along with a `ACCESS_MASK` that includes all the possible `ACCESS_MASK`s the program needs.

3.2.3 Enforcement Mode. In the EnfM, MOMS references an OAL to determine if an operation should occur. The OAL consists of the objects in the ORL, with more or fewer objects, or reduced or increased access rights. The decision of whether to restrict, deny, or allow, an operation depends on the process that receives the handle and, for handle duplication, the process that the handle originates from. If the process that receives the handle is not in an APL, the operation is allowed, since the process is not being sandboxed. For create operations, if the object has a name, the operation is restricted to those that are in the OAL of the program that the process receiving the handle belongs to. If the object does not have a name, the operation is allowed. For duplicate operations, the operation is allowed in the following scenarios:

1. the process the handle originates from and the process that receives the handle are in the APL of the same program,
2. if the object has a name, the operation is restricted to those in the OAL of the program,
3. if the object does not have a name, the handle originates from the parent process of the process that receives the handle or if the process that the handle originates from is a system process, such as *explorer.exe* or *svchost.exe*.

When an operation is restricted to the objects in an OAL, if the name of the object is in the OAL, the `ACCESS_MASK` is limited by the `ACCESS_MASK` in the OAL; otherwise, all rights in the `ACCESS_MASK` are removed. The EnfM also ensures a process does not access an object it should not have access to through an object it should, such as an object that is a hard, or symbolic, link.

This algorithm ensures each program does not access a named object it should not and that unnamed objects do not cross the program boundary. Unnamed objects are

allowed to be duplicated from a process to its child process, or from a system process, without restriction because this is required for normal operation.

3.3 System Services

MOMS provides two main services: generation of the OAL and the enforcement of it. The sandbox administrator generates the OAL based on the ORL produced by the RGM. The OALs used for these tests are minimally altered versions of the ORLs generated by the RGM. Minimal alterations consist of changes such as allowing access to temporary directories instead of specific files in those directories. The EnfM enforces the OAL, therefore it depends directly on the OAL. The two services will not be tested separately, the combined effect of them will be tested. The two services are tested by running an exploit against a sandboxed program and recording whether the associated payload succeeds or fails to execute. The outcomes of the test for the two services are:

Success: the payload is not able to execute

Failure: the payload is able to execute

3.4 Workload

To test MOMS, a set of programs will be used, along with associated program tasks. The performance metrics, especially the performance metrics measuring efficiency, depend on the particular program and program task being used.

Program The performance of a program depends on the details of that program, such as the amount of input and output operations it does and the amount of memory it uses. More specific to MOMS, the number of total handle create and duplicate operations and the number of unique objects the program requests a handle to affects the performance of the program, since the higher either of them are, the more work MOMS must do.

Program Task The performance of an individual program not only depends on the individual program, but also on the task it is doing, such as the document it is opening.

3.5 Performance Metrics

There are two main performance metrics that measure the performance of MOMS and determine whether the goals for the system have been achieved:

Efficiency of the RGM and the EnfM Determined by the execution time or execution speed of a set of programs with and without the RGM,

Effectiveness of the RGM and the EnfM Determined by the success or failure of the payload of a associated with an exploit.

3.6 System Parameters

The two CUTs have many system parameters in common and are therefore listed together.

Number of Processor Cores: This parameter includes the total number of cores: whether they are on different physical processors within the same computer, on the same physical processor, or some combination. The number of processor cores affects the performance of multi-threaded programs and the performance of programs that run at the same time as other programs.

Processor Speed (Clock Frequency): The processor speed affects the speed that programs run at, that is the higher the processor speed, the lower the execution time of the program.

Memory Size: Memory size is important because access lists can get large enough that they do not fit in available memory and may need to be stored on the hard drive. The

more memory the system has the less the processor must go to the hard drive to retrieve the data.

Memory Speed: Memory Speed is important because faster memory can respond to processor requests quicker and therefore the faster the program runs.

Hard Drive Speed: There are two types of drives: hard drives (HDs) and solid state drives (SSDs). The speed of HDs are reported as the number of revolutions per minute they make. The speed of SSDs are reported as how much data they can read or write per second. The HD speed can have an important role if the processor has to wait on the HD for data.

3.7 System Factors and Levels

Since the RGM and the EnfM are very similar and they will run on similar machines, the same factors and levels pertain to both of them or neither of them. Table 3.1 contains the system factors that are considered when testing MOMS and the corresponding levels that are considered when evaluating MOMS.

The *Program* being ran and the *Program Task* are included as factors because they can affect the efficiency of the system. The levels chosen for the *Program* being ran are chosen to represent the kinds of programs someone would use on a daily basis. Benchmarks are not used so that the effectiveness of MOMS, at preventing the payload of an exploit from successfully executing, can be tested with the same programs that are used for efficiency testing. The levels chosen for the *Program Task* are chosen for each *Program* in order to exercise some of the functionality of the program. For *7-Zip 9.20*, the first task is performing compression and the second task is performing decompression. Each task is opening a web page, for *Internet Explorer 8*. For the rest of the programs, each task is opening an appropriate document.

The *Number of Processor Cores* is included because access checks could become a bottle neck if the system has to wait for too many checks and the more cores there are the less of a concern this is. The levels, one and two cores, are chosen. *Memory Size* is often an important factor for system performance, which is important for the EnfM. Two levels are chosen for the memory size: 2 GBs because it is a common size for basic systems running *Windows 7* and 3 GBs because it is the maximum available on the computer used for testing. The number of cores, and the available memory are controlled with the Windows boot options, the underlying hardware of a two core processor and 4 GBs of installed physical memory did not change.

The *Processor Speed* is not a factor because it does not have a significant impact when compared to the micro-architecture of the processor and the number of cores. In addition, *Memory Speed* is not a factor because there is not a significant enough a difference between the speeds found in most workplace computers for it to be an important factor. Due to the relatively small size of MOMS, *HD Speed* should not have a significant impact on the operation of MOMS.

3.8 Evaluation Techniques

Two different evaluation techniques are used to fully evaluate MOMS: measurement of a real system and analytical modeling.

Measurement of a real system evaluates the performance of MOMS and its effectiveness against current attacks. To evaluate the performance of MOMS, the programs in Table 3.1 are used. There are 250 repetitions of each task for each program, which should meet the statistical requirements in Section 3.9. Each program is run a total of 250 times, because when each program was run for 100 repetitions, initially, the resulting data was not normally distributed, so they were run an additional 150 times in an attempt to get more normally distributed data.

Table 3.1: System Factors and Levels for MOMS

System Factor	**Level**
Program	7-Zip 9.20
	Acrobat Reader 9.0
	Internet Explorer 8
	Microsoft Excel 2007
	Microsoft Power Point 2007
	Microsoft Word 2007
Program Task	Task 1
	Task 2
Number of Processor Cores	1 core
	2 cores
Memory Size	2 GB
	3 GB

A single test run of a program consists of running the program for each task. The program opens the first task, an appropriate document or web page, and closes it, then it opens and closes the second task. The execution time is the total amount of time that passes from when the program is told to open the task to when the program has exited. The execution time for the first task and the second task are recorded as the performance metrics for the program. An exception, to evaluate *7-Zip 9.20*, the built in benchmark is used. The built in benchmark measures the compression and decompression speeds in KB/s and runs for the number of desired repetitions. The compression speed is recorded as the performance metric, execution speed, for the first task and the decompression speed is recorded as the performance metric for the second task.

A complete test run of a program consists of running the single test run for the program for the number of desired repetitions plus one; the first single test run is ignored, since there is too much variability in the performance metrics for the first run. A system test run consists of running the complete test run of each program in succession. Each system test run is run without and with the RGM.

The evaluation of the RGM and the EnfM is in two parts: evaluation of its effectiveness against current attacks and an analysis of its effectiveness against future attacks designed to circumvent it. To evaluate its effectiveness against current attacks, a set of exploits is used, along with appropriate payloads. Malware is composed of two main components: the exploit, which uses a vulnerability in a program to execute some arbitrary code, and the payload, which is the arbitrary code an exploit executes. The OALs are tuned as if they are going to be deployed in a production environment. The exploits are then run and whether the associated payload successfully executes is recorded. To evaluate the effectiveness of the EnfM against future attacks designed to circumvent it, an analysis is done on the EnfM to determine potential vulnerabilities that may be exploited.

3.9 Experimental Design

The experimental design evaluates the efficiency and effectiveness of the RGM and EnfM. The efficiency measurements depend on the execution time of the RGM and the EnfM which is a probabilistic measurement, so some repetitions are required. In addition, the factors related to the efficiency have a lot of interaction and there are not many factors/levels, so a full factorial design is appropriate. Since the timing of the CUTs are not critical (they have to be fast enough so people will use them) a confidence level of 90% is appropriate. The tests to determine the effectiveness of the RGM and EnfM will be reported without statistical analysis, since they are deterministic results, so no repetitions are required, and there are not many results.

3.10 Summary

MOMS consists of two main components, RGM and EnfM, and two secondary components, Administration Computer and Production Computer. The RGM runs on the Administration Computer and the EnfM runs on the Production Computer. There are two metrics used to evaluate this system: the efficiency of the RGM on the Administration Computer and the combined accuracy of the RGM and the EnfM. To evaluate the system based on these metrics a set of representative programs will be run on different configurations of the system to determine the efficiency of MOMS and the effectiveness of MOMS against current threats. To evaluate the accuracy of the EnfM against future threats designed to circumvent it, an analytical evaluation approach is used. The confidence level for the efficiencies only needs to be 90%, since the system only needs to be fast enough that people will use it.

4 Results and Discussion

This chapter determines the impact the RGM has on the performance of the system and the effect the hardware configuration, the *Number of Processor Cores* and the *Memory Size* factors, has on performance. In addition, the effectiveness of MOMS in sandboxing programs will be determined in this chapter as well.

To determine the effect MOMS has on the performance of the system, the performance metrics for each factor level combination without the RGM, the base system, will be compared with the performance metrics for the same factor level combination with the RGM. For each comparison, the *Wilcoxon signed rank test* will be used, which is the *Mann-Whitney U test* for paired groups. The paired groups version of the *Mann-Whitney U test* is used, because the performance data for with and without MOMS is dependent on each other; the performance of the system with MOMS depends on the base system, since the better or worse the base system performs, the better or worse the system with MOMS performs [26, p. 165]. The *Wilcoxon signed rank test*, a non-parametric test (it does not assume the data conforms to a particular distribution), will be used instead of a parametric test like the *t-test*, because the underlying data does not have a consistent underlying distribution.

To determine the effect the hardware configuration has on performance, for each program configuration (the *Program* and the *Program Task* factors), the performance metrics associated with each hardware configuration will be compared pair-wise. They will be compared with the *Wilcoxon signed rank test*, for the same reasons as above. A pair-wise comparison of each hardware configuration is conducted, instead of a linear model, to maintain consistency, since not all of the data conforms to an underlying distribution and therefore a linear model cannot be done with that data.

Before the impact the RGM and the effect the hardware configuration has on performance is determined, the underlying data will be examined in order to determine if

the data has any internal relationships, that is departures from randomness, and to determine how the data is distributed.

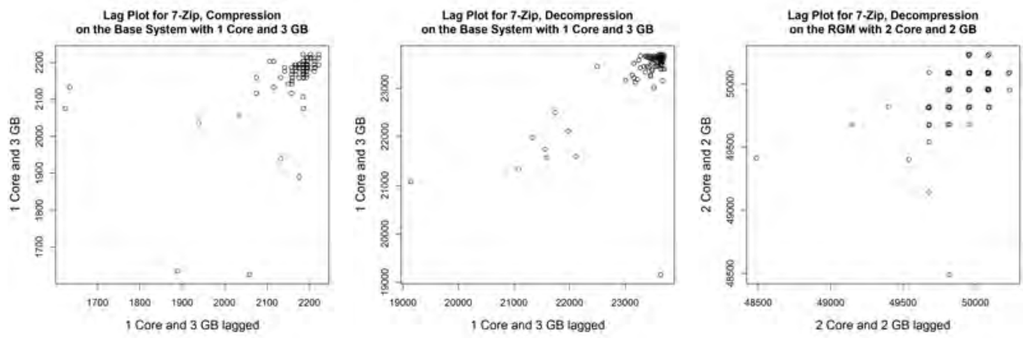
4.1 Internal Relationships of the Data

To determine whether a sample set has an internal relationship an autocorrelation plot will be used. When the autocorrelation plot indicates the sample set may be internally related, a lag plot will be used to view what the relationship is. Autocorrelation plots graph the autocorrelation value of a sample shifted by multiple values, k. The autocorrelation value measures the amount of correlation between a sample set and the shifted sample set. It is a continuous value from 1 to -1, where 1 indicates strong positive correlation, 0 indicates no correlation, and -1 indicates strong negative correlation. The closer the autocorrelation value is to 0, the more random the sample is. Lag plots visually represent the autocorrelation plot for $k = 1$. Lag Plots also show the existence of outliers.

The autocorrelation plots for each program is in Appendix A. For each program, the autocorrelation plots show that the data for most of the configurations are random. For the configurations of a program that are not random, lag plots are used to determine the relationship of the data.

All of the configurations of *7-Zip* is random, as shown in Figure A.1, except for three: the base system with one core and 3GBs for compression and decompression and the RGM with two cores and 2GBs for decompression. The lag plots in Figure 4.1 show a linear relationship for the data points, centralized in the upper-right corner, and it shows several outliers.

Figure A.2 shows that the underlying data for each configuration of *Acrobat Reader* is random. However, the underlying data for three configurations of the *Excel* Spreadsheet 1 (the base system with one and two cores, and 2 GBs memory, and for the RGM with 2

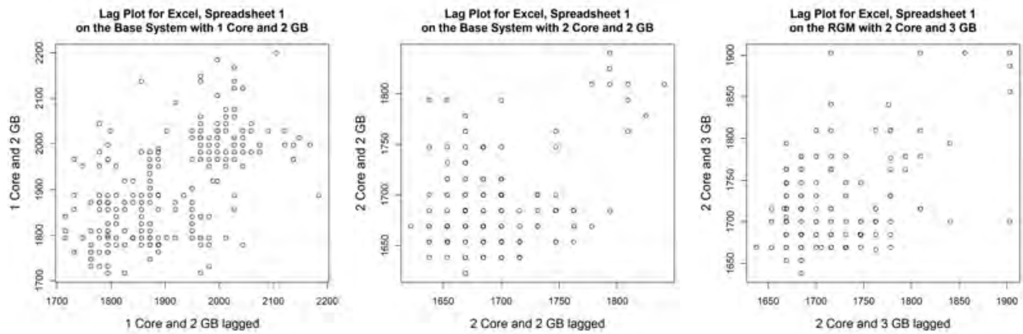


(a) Base System - Compression: 1 Core, 3 GB

(b) Base System - Decompression: 1 Core, 3 GB

(c) RGM - Decompression: 2 Cores, 2 GB

Figure 4.1: Lag Plots for 7-Zip

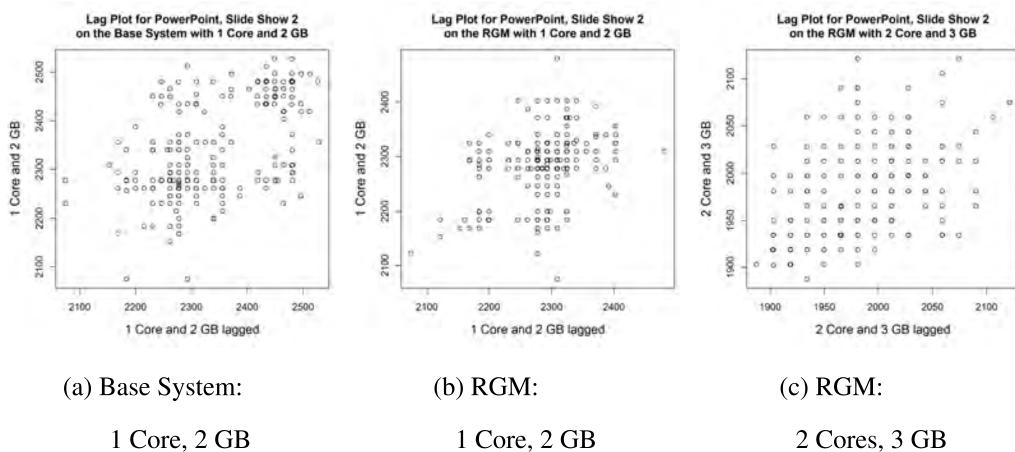


(a) Base System: 1 Core, 2 GB

(b) Base System: 2 Cores, 2 GB

(c) RGM: 2 Cores, 2 GB

Figure 4.2: Lag Plots for Excel, Spreadsheet 1



(a) Base System: 1 Core, 2 GB

(b) RGM: 1 Core, 2 GB

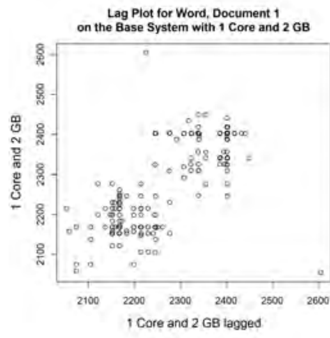
(c) RGM: 2 Cores, 3 GB

Figure 4.3: Lag Plots for PowerPoint, Slide Show 2

cores and 3 GBs memory) is not random, as Figure A.3a shows. As shown in Figure 4.2, the samples have a widely varying slightly linear relationship.

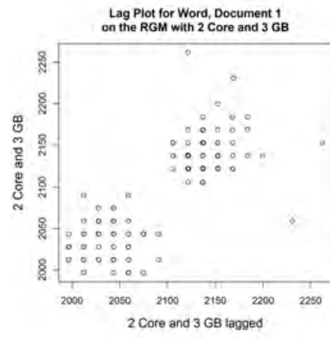
The underlying data for *Internet Explorer*, shown in Figure A.4, in all configurations, is random. Figure A.5b shows that the underlying data for Slide 2 for *PowerPoint* is not random for three of the configurations: the base system with one core and 2 GBs of memory, the RGM with one core and 2 GBs of memory, and the RGM with 2 cores and 3 GBs of memory. Figure 4.3 shows that the samples have a widely varying slightly linear relationship as is the case for the *Excel* Spreadsheet 1.

Finally, Figure A.6 shows that the data for the base system with one core and 2 GBs of memory, and for the RGM with two cores and 3 GBs of memory, for both *Word* documents, may be non-random. In addition, it shows that the base system, with one core and 3 GBs of memory, for the *Word* document 2 may also have some non-randomness. Figures 4.4a to 4.4d show that the samples have a linear relationship and are therefore not random. Figure 4.4e shows that the samples for the base system, with one core and 3 GBs of memory, for the *Word* document 2, is random, with the exception of some outliers.



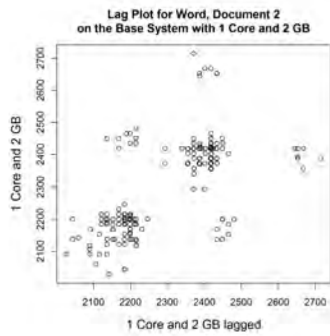
(a) Base System - Document 1:

1 Core, 2 GB



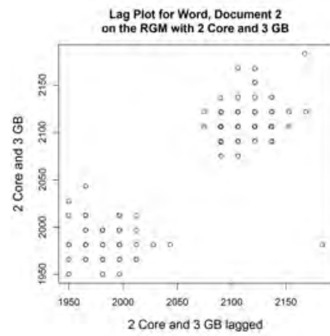
(b) RGM - Document 1:

2 Cores, 3 GB



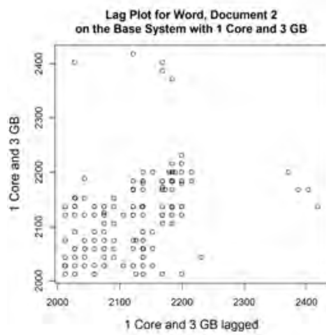
(c) Base System - Document 2:

1 Core, 2 GB



(d) RGM - Document 2:

2 Cores, 3 GB



(e) Base System - Document 2:

1 Core, 3 GB

Figure 4.4: Lag Plots for Word

4.2 Determine Data Distribution

Distribution plots will be used to show how a sample set is distributed. Distribution plots consist of a histogram of the data, a kernel density plot, and the kernel density plot for a normal distribution superimposed on the same graph. The histogram groups the values of the sample set into a number of bins and graphs the probability density of each bin, which is based on the number of data points in each bin. The kernel density plot, in this case, serves as a continuous version of the histogram. Both the histogram and the kernel density plot show the shape of the sample set. The kernel density plot for a normal distribution shows the shape of a normal distribution based on the sample set, the mean and standard deviation of the normal distribution are set to the mean and standard deviation of the sample set [26, p. 128-132].

The sample sets are compared to the normal distribution because it is a common distribution and will provide a good basis for comparison. To determine if a sample set is normally distributed, the kernel density plot of the sample set and the normal distribution are compared; if the two kernel density plots are matched closely enough, the sample set is considered normally distributed.

Appendix B contains the distribution plots for each configuration for each program. While the samples for some of the configurations, such as the base system with one core and 2 GBs of memory for *Word* document 1, shown in Figure B.6a, are approximately normally distributed, not many of them are. Moreover, some of the program and task pairs have no configuration that is normally distributed, such as *Acrobat Reader* PDF 1, as Figure B.2a shows. Furthermore, there are many different types of distributions and some of the sample sets do not conform to any standard distribution, such as *Word* document 2, as shown in Figure B.6b.

4.3 Effect of Factors on Performance

To determine which of the hardware configuration factors have a strong impact on the performance of the system, each hardware configuration is compared pair-wise, for each program configuration. To visualize the effect the hardware configuration factors have on performance, for each program configuration pair, the box plots representing the sample sets of each hardware configuration are graphed next to a graph of the difference between each pair-wise hardware configuration. If the calculated difference between a pair of hardware configurations is not statistically significant (its *p-value* is greater than 0.1), then it is graphed as being zero on the the difference graph. The graphs for each program are in Appendix C.

The graphs for *7-Zip*, Figure C.1, show the effect the different hardware configuration factors have on the performance of *7-Zip* compression and decompression, on the base system and with the RGM. The graphs show that the only time the performance metric, compression or decompression speed, changes by any practically significant amount is when the number of cores change. A practically significant performance change is one that a user may be able to notice; a performance change that is not practically significant is one that a user will not notice.

No practically significant performance metric changes occurred for *Acrobat Reader* PDF 1, as shown in Figure C.2a. Figures C.2b to C.5 show a negligible difference (a practically significant difference a user probably will not notice), for *Acrobat Reader* PDF 2 and the other programs, except *Word*, when the number of cores differ, but show no practical difference for changes in memory size. *Word* shows a slightly higher impact to performance from memory size differences than for the previous program configuration pairs, as Figure C.6 shows.

Based on the aggregation of the above results, the number of cores has an impact on performance, while the amount of memory does not, regardless of whether MOMS is

running or not. Even though the number of cores has an effect on performance, the impact is negligible, except for *7-Zip*, since the maximum difference in execution time is about 400 milliseconds. The impact that the number of cores has on the compression and decompression speed of *7-Zip* is non-negligible, especially for large files, at about 2 MB/s and 26 MB/s improvement, respectively. The probable reason that the memory size does not have a practical impact on performance is that the test programs do not use enough memory to be limited by the lower memory size limit, therefore they do not use the extra memory, even though it is available. The memory size may have more of an impact when other programs are used to test the system or if enough other tasks are going on. Therefore, when deciding what the hardware configuration should be for a computer running MOMS, MOMS should be tested with a workload that more accurately reflects the environment of the average user; however, for the particular test programs used and the, tasks they performed, the hardware configuration has a negligible impact on performance, except for *7-Zip*.

4.4 Effect of Magnesium Object Manager Sandbox on Peformance

Now that the effect of the hardware configuration factors have been determined, the effect MOMS has on the performance, regardless of the workload or other factors, will be determined. This analysis will be conducted in two parts, visually and through statistical analysis. The visual analysis consists of box plots representing the performance data of the system with and without MOMS, for each combination of factor level and workload; this allows one to get a feeling of the relationship. The statistical analysis is a *Wilcoxon signed rank test* to determine whether the perceived relationship is statistically significant, and if it is whether there is a practical difference. Appendix D contains the box plots and the statistical analysis for each hardware and program configuration pair.

For the hardware configuration of one core and 2 GBs of memory, the box plots, shown in Figure D.1, indicate that the RGM mode decreases the performance of *Internet*

Explorer for both web pages and actually increases the performance of *Word* for the second document and possibly for the first document. Since, the RGM adds additional processing to the system and takes no steps to improve performance, such as performing additional caching, performance cannot increase with the RGM running. Therefore, the perceived performance increase is probably due to the system experiencing more overhead when data was being gathered on the base system than on the system with the RGM running. The additional overhead could come from sources such as the system responding to increased network traffic or other processes conducting background work, even though steps are taken to reduce this. The box plots indicate that for all the other program configurations there is no difference in the performance of the base system and the system with the RGM running.

The statistical analysis for the one core and 2 GBs of memory hardware configuration, shown in Table D.1, shows that *Internet Explorer* does decrease performance for both web pages and that *Word* does increase performance for both documents. All the other program configurations either have no statistically significance or no practical difference. The impact that the RGM has on *Internet Explorer* for the two web pages, and on *Word* for the two documents, is negligible, with a maximum impact of less than 400 milliseconds.

For one core and 3 GBs of memory, Figure D.2 indicates that the RGM decreases the performance of *Internet Explorer* for both web pages, possibly decreases performance for both slide shows for *PowerPoint*, and possibly increases the performance of *Word* on the second document. The apparent increase in performance of *Word* for the second document is probably due to the same reasons as above. Table D.2 shows that the RGM decreases performance negligibly for both web pages on *Internet Explorer*, but does not impact performance for the other programs, either statistically or practically significantly.

The RGM decreases performance for the first web page on *Internet Explorer*, for both slide shows on *PowerPoint*, and on *Word* for both documents, for the hardware configuration of two cores and 2 GBs of memory, as Figure D.3 shows. It also indicates that the performance of *Internet Explorer*, for the second web page, may increase with the RGM running. Table D.3 shows that MOMS negligibly decreases the performance of *Internet Explorer* for the first web page, but does not impact performance, statistically or practically, for any of the other program configuration pairs.

Finally, for the two cores and 3 GBs of memory hardware configuration, Figure D.4 indicates that the RGM decreases performance for the first PDF on *Acrobat Reader*, for the first web page on *Internet Explorer*, and for both documents on *Excel*, *PowerPoint* and *Word*. Table D.4 shows that the first web page on *Internet Explorer* has a negligible performance decrease due to the RGM, as does both documents for *Word*. It shows that there is no statistically or practically significant performance impact for the other program configuration pairs.

As shown in Appendix D and discussed above, and as summarized in Table 4.1 (the statistically insignificant values have been set to zero), MOMS has a negligible impact to performance.

4.5 Effectiveness of Magnesium Object Manager Sandbox

A set of exploits are run against the test programs to evaluate the effect the EnfM has on the success of the associated payload. Since MOMS is not intended to protect programs from compromises, but rather to protect the system from compromised programs, an exploit is successful if its payload executes successfully. If the payload successfully executes, it indicates that the sandboxed program gained access to more objects than it is suppose to.

Table 4.1: Summary of the Impact the RGM has on Program Performance

Program	**Document**	**1 Core 2 GB**	**1 Core 3 GB**	**2 Cores 2 GB**	**2 Cores 3 GB**
7-Zip	Compression	-1 KB/s	0 KB/s	0 KB/s	0 KB/s
	Decompression	0 KB/s	0 KB/s	0 KB/s	0 KB/s
Acrobat Reader	PDF 1	-0.04 ms	-0.65 ms	-0.15 ms	-15.51 ms
	PDF 2	-10.82 ms	0 ms	-15.51 ms	-0.05 ms
Internet Explorer	Web Page 1	-374.4 ms	-358.94 ms	-265.22 ms	-280.79 ms
	Web Page 2	-73.53 ms	-73.61 ms	13.24 ms	0 ms
Excel	Spreadsheet 1	15.62 ms	-31.17 ms	-15.47 ms	-31.19 ms
	Spreadsheet 2	0 ms	-31.21 ms	-15.57 ms	-31.13 ms
PowerPoint	Slide Show 1	0 ms	-31.23 ms	-31.19 ms	-46.95 ms
	Slide Show 2	16.29 ms	-15.6 ms	-31.25 ms	-62.46 ms
Word	Document 1	77.97 ms	-15.6 ms	-41.81 ms	-124.71 ms
	Document 2	125.03 ms	46.84 ms	-31.16 ms	-140.3 ms

Exploits for each test program were searched for. Of the exploits found, only the ones that are able to successfully execute their payload (many are able to compromise the program but are unable to execute a payload, most likely due to other protections provided by Windows 7 32-bit) are used to determine the effectiveness of the EnfM. Table 4.2 shows the EnfM stops all the payloads from executing, which indicates the EnfM is effective at preventing exploit payloads from causing damage to the system; however, this does not mean that the EnfM is effective at preventing all exploit payloads from causing damage to the system.

Table 4.2: Successfullnes of Exploits with the EnfM Running

Target Program	**Exploit**	**Payload Executed**
Internet Explorer 8	IE Unsafe Scripting Misconfiguration [31]	No
	Sun Java Runtime New Plugin docbase Buffer Overflow [24]	No
	Internet Explorer CSS Recursive Import Use After Free [37]	No
	MS11-050 IE mshtml!CObjectElement Use After Free [15]	No
Adobe Acrobat Reader 9.0	Adobe CoolType SING Table “uniqueName” Stack Buffer Overflow [23]	No
	Escape From PDF [46]	No

Even though the EnfM is effective at preventing the exploits in Table 4.2, it does not mean that it can prevent all exploit payloads. Some possible ways the EnfM could be circumvented are through file system hard links, too generalized OALs, and non-specific filenames.

File system hard links are a feature of the NTFS file system that essentially gives a file two or more names [20]. This can lead to a compromise of the rest of the system if a file the sandboxed program should not have access to has a hard link to it made that has the name of a file the sandboxed program should have access to; in other words, a file a sandboxed program should not have access is given an additional file system level name of a file the sandboxed program can access to, thereby giving the sandboxed program access to the file it should not be able to access. To prevent this, the EnfM could be improved to not allow access to any file that had a hard link made to it (can be accomplished through the Windows API [18]), or if a file does have hard links to it, access can be allowed only if all of the hard links are in the OAL (can be done through examining the NTFS file attributes [32, 8], such as by using Windows utilities [17, 48]). Symbolic links are not a problem because they require administrative access to create and the symbolic links are resolved when the name lookup takes place, therefore the EnfM would not see the symbolic name, just the actual name of the file.[14]

As is, the EnfM mode can allow access to objects just based on the beginning of the name of the object. This is done so that the program can function without having to list every object the program can access, even if the full name may not be known (such as cached data in *Internet Explorer*) or is partially random. If the OAL is constructed in such a way that one of these generalized entries gives the sandboxed program access to an object it should not have access to, then the EnfM could be ineffective against the payload of an exploit. This risk can be mitigated by allowing the OAL entries to be more specific, such as through regular expression matching, allowing the administrator to specify the form of the name, rather than just the beginning of the name that is constant.

Currently, MOMS looks for the main process by its short name, the name that shows up in *Windows Task Manager*. Any executable on the system could be named the same as one of the main processes. While this does not create a vulnerability, since a program that

has the same name of a main process would just be put into the sandbox of the main process, this could cause problems as MOMS is further developed. Therefore, MOMS should look for main processes based on the full path of the executable, rather than just the short name [52, 13].

5 Conclusion and Future Work

MOMS is created as an alternative to the existing sandboxes that intercept system calls in order to sandbox programs, and it is similar to sandboxes such as *SELinux* and *FreeBSD Jails*. MOMS is implemented using the function callback functionality provided by the Windows OM, which is the component that manages all objects and distributes handles to those objects to user mode programs. Using this method, the RGM of MOMS logs the objects that a sandboxed program accesses, the sandbox administrator then modifies this list to create the OAL, and then the EnfM enforces the OAL

Once MOMS is implemented, a set of programs are used to test its efficiency and effectiveness. Each test program had two different tasks to perform, such as opening an appropriate document. Each program and task is run with and without the RGM, with 2 or 3 GBs of memory, and with one or two processor cores. The RGM is used to also indicate the performance of the EnfM. The performance metrics are analyzed in two ways. To determine whether the different hardware configurations, memory size and number of processor cores, significantly impacted the performance of MOMS, the performance data for each hardware configuration is compared, for each combination of program, program task, and whether the RGM is running. From these comparisons, it is determined that the number of processor cores generally affected performance much more than the amount of memory, which is probably due to the test programs not using enough memory to benefit from the increased memory; however, the performance is not affected enough to have a significant practical effect, especially for the memory size. To determine the impact the RGM has on performance, the performance metrics for each program, program task, and hardware configuration combination, with and without the RGM running are compared. This comparison showed that there is a statistically significant impact to performance when programs are run with the RGM, but the difference is not practically significant.

These results have limited application, since the programs, and the program tasks are limited and not chosen at random; however, they do indicate that the performance impact of the RGM is minimal, regardless of the hardware configuration. To make these results more applicable, more programs and program tasks must be tested in a similar manner as above. The programs and program tasks should also be chosen at random from programs and tasks that represent the normal workload of a user. In addition, the performance of the programs when the system is running several programs, such as what a normal user would run, should be explored.

In addition to the performance impact of MOMS being tested, the effectiveness of MOMS is determined by testing exploits to a subset of the above test programs and recording whether the payload associated with each exploit executed successfully. MOMS is successful in preventing the payloads of all the tested exploits from executing. Again, these results have a limited applicability, but they do show that further testing, with a wider range of exploits, is warranted. Furthermore, MOMS is analyzed to identify possible vulnerabilities that could be exploited in order to limit its effectiveness. While there are some vulnerabilities in the current implementation of MOMS, they are all straight forward to fix.

5.1 Alternative Sandbox Methods

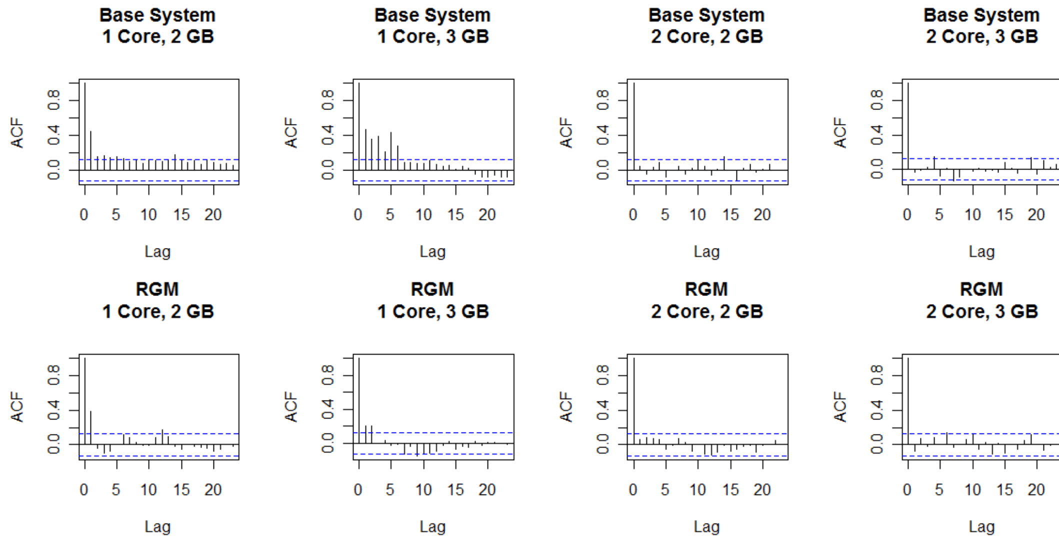
5.1.1 Automatic Program Confinement to Private Namespaces. Currently MOMS only limits the effect a sandboxed program can have on the system, but it would be beneficial to explore ways to limit the effect other programs can have on a sandboxed program. One way to do this is to place objects a sandboxed process accesses into a private namespace. Private namespaces are a feature of the OM that allow a program to limit the processes that can access its objects [34]. While this feature is intended for programs to place themselves into a private namespace, it may be possible to extend this feature to allow arbitrary programs into their own private namespace, without rewriting

the program. If it is possible it would be an effective way to limit the actions other programs can take on the sandboxed program.

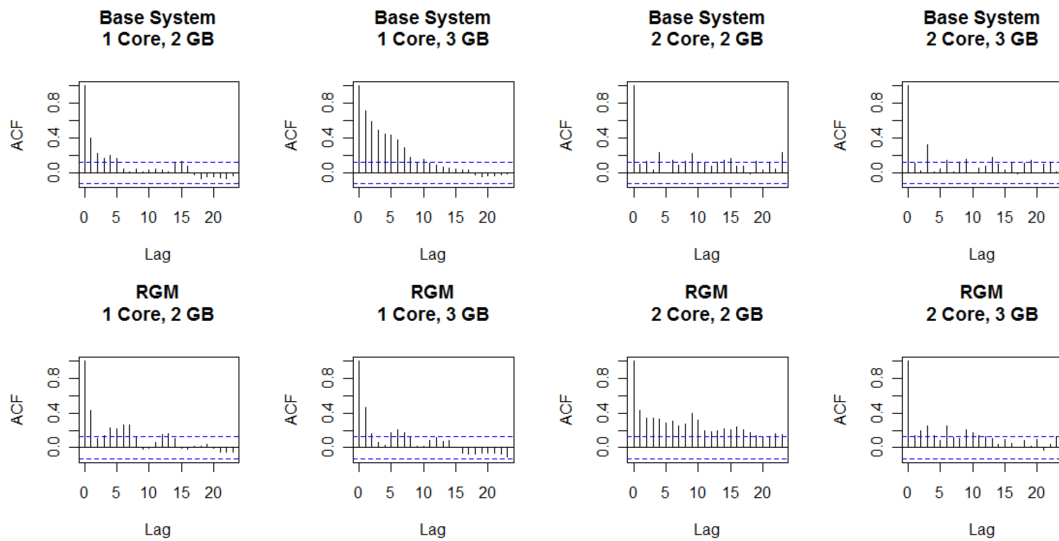
5.1.2 Automatically Assign Processes to Jobs. An alternative method to restrict the objects a process can access is to create Job objects that have only the rights for a program to run, and then assign processes of the program to the Job. This method would provide similar protections as the MOMS does, but it would rely entirely on official functionality (currently the *SupportsObjectCallbacks* field must be altered for each object type) and it may be more efficient. This method could also be extended to limit the ways processes can interact with sandboxed programs by assigning them to a Job that does not allow them access to the objects that the sandboxed programs access.

In its current state, MOMS is a good start to exploring other sandboxing methods on Windows; however, more work is needed to expand this research and to determine the best way to sandbox programs on Windows.

Appendix A: Autocorrelation Plots

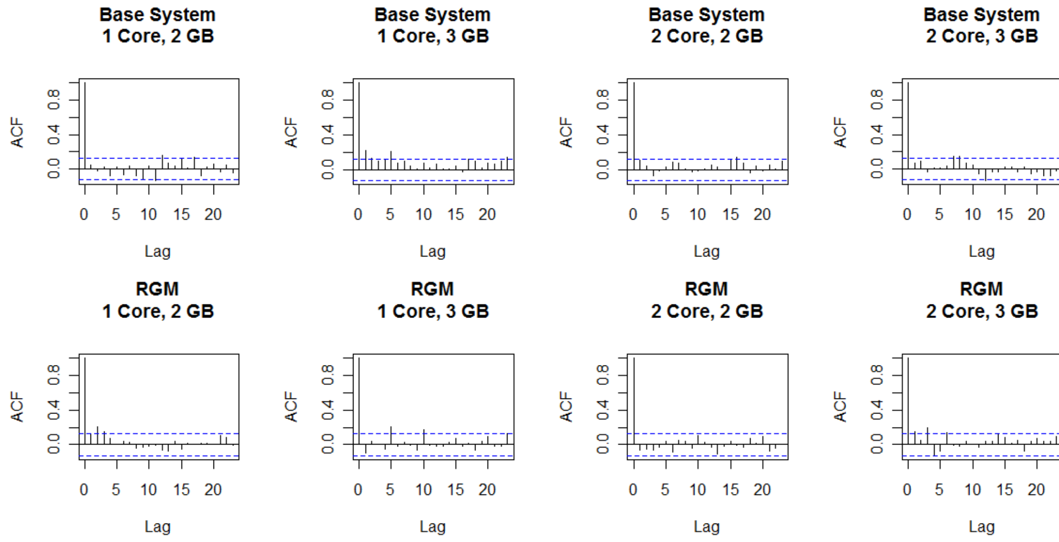


(a) Compression

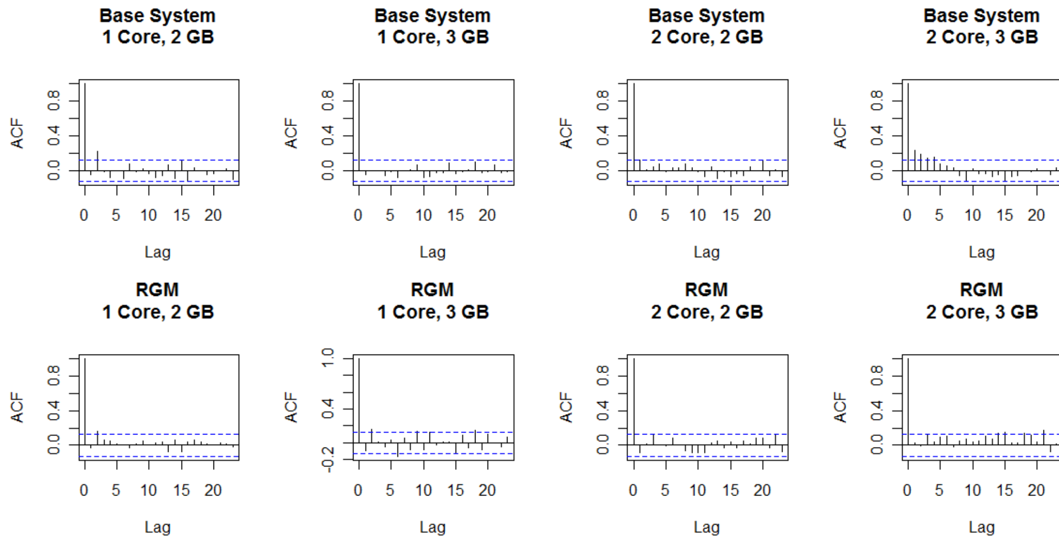


(b) Decompression

Figure A.1: Autocorrelation Plots for 7-Zip

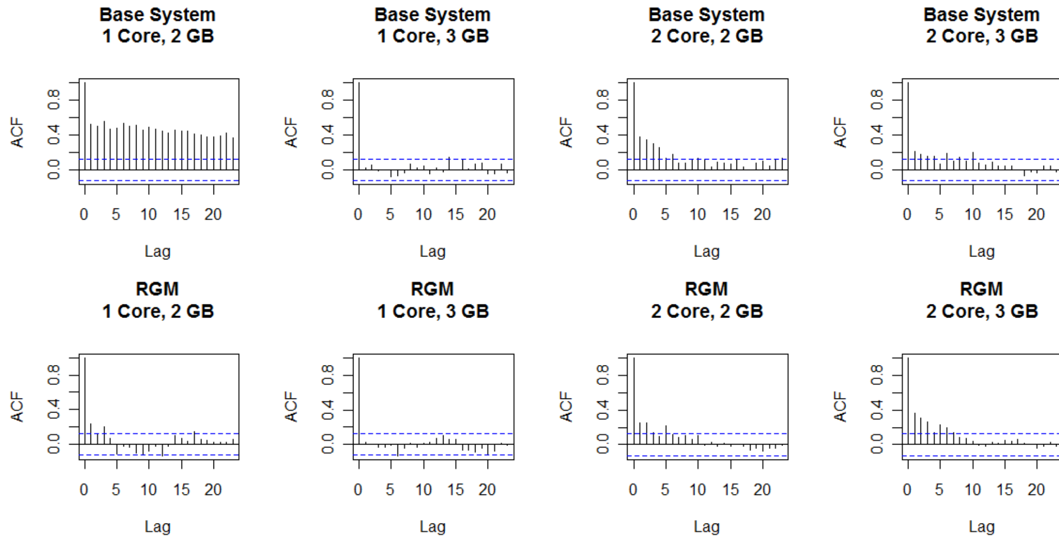


(a) PDF 1

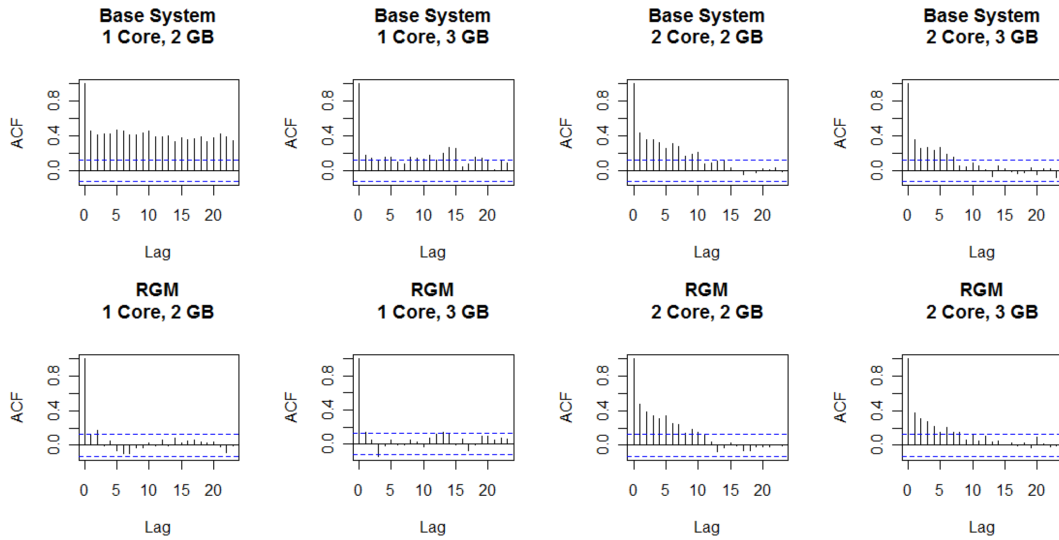


(b) PDF 2

Figure A.2: Autocorrelation Plots for Acrobat Reader

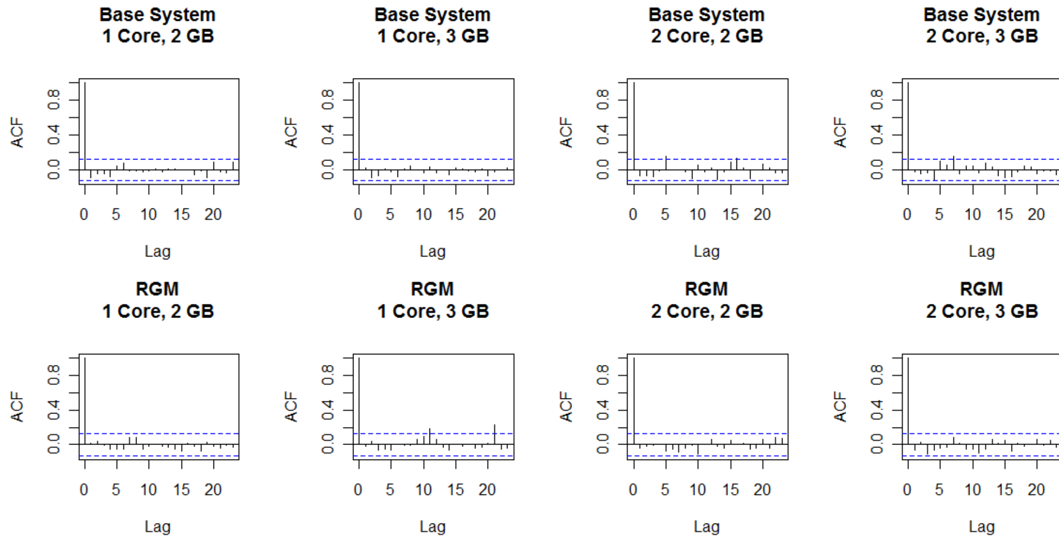


(a) Spreadsheet 1

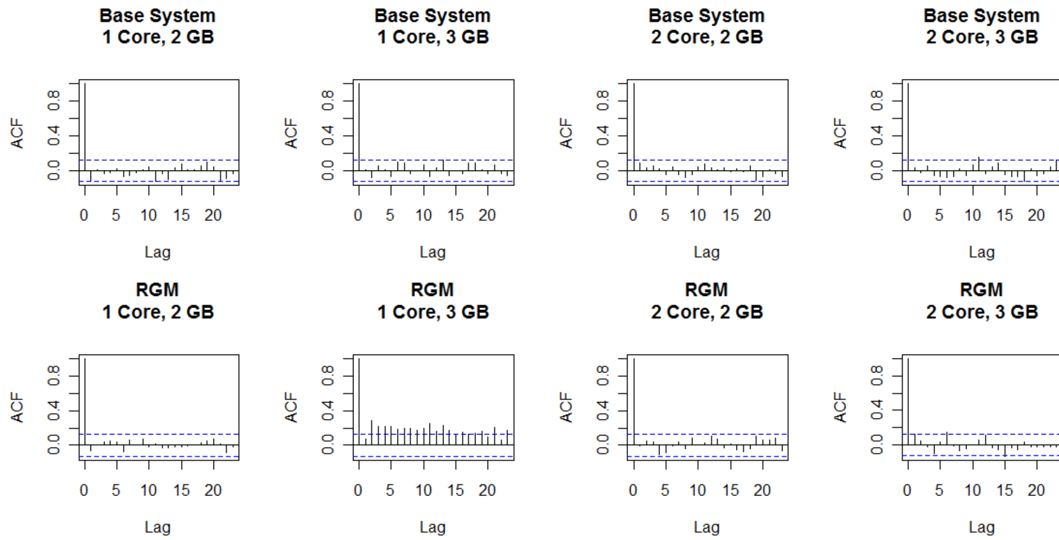


(b) Spreadsheet 2

Figure A.3: Autocorrelation Plots for Excel

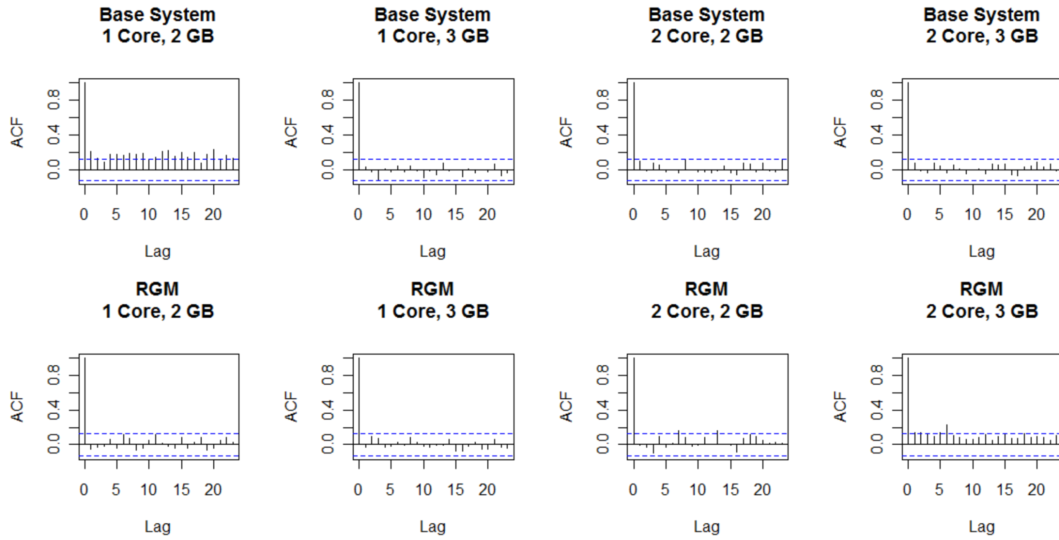


(a) Web Page 1

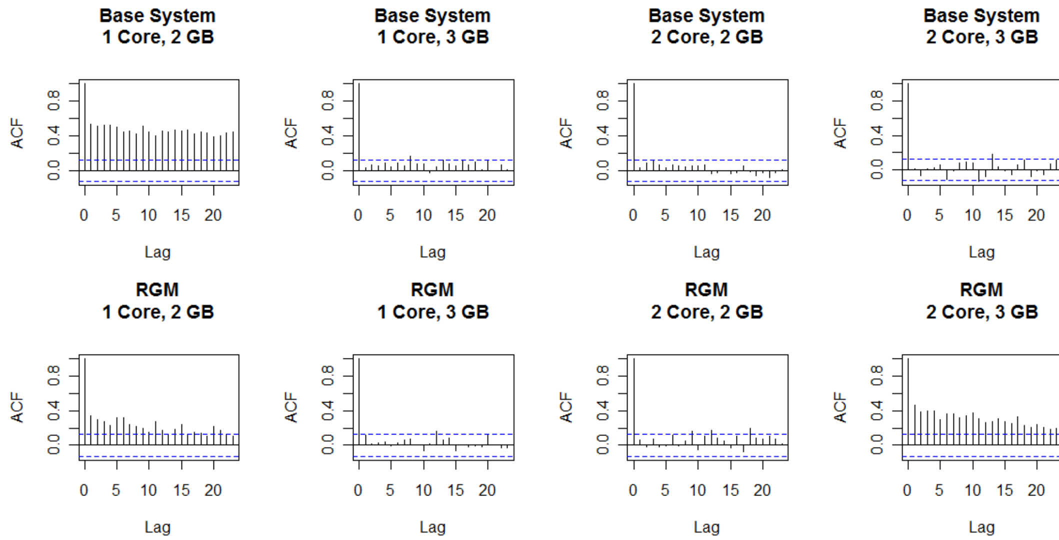


(b) Web Page 2

Figure A.4: Autocorrelation Plots for Internet Explorer

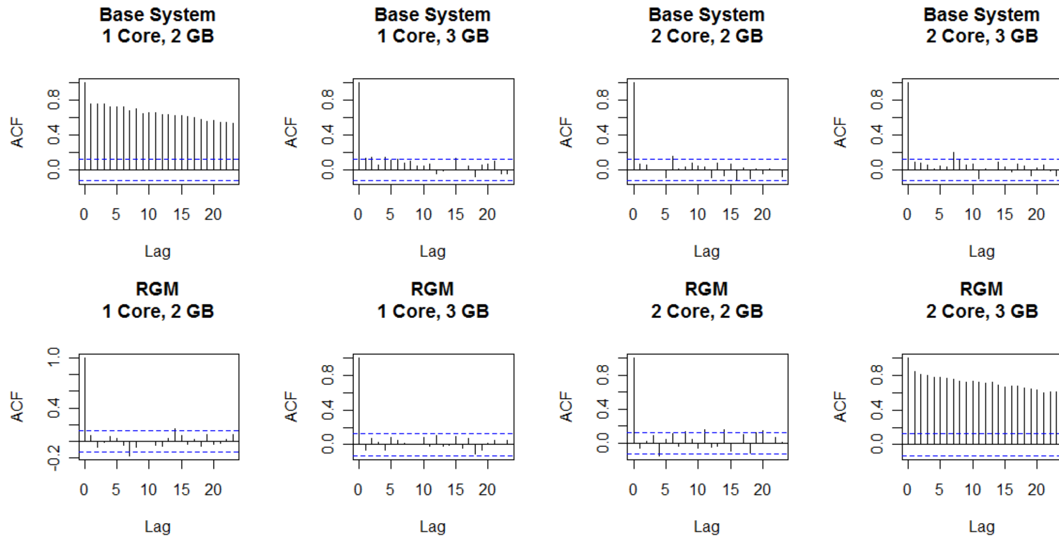


(a) Slide Show 1

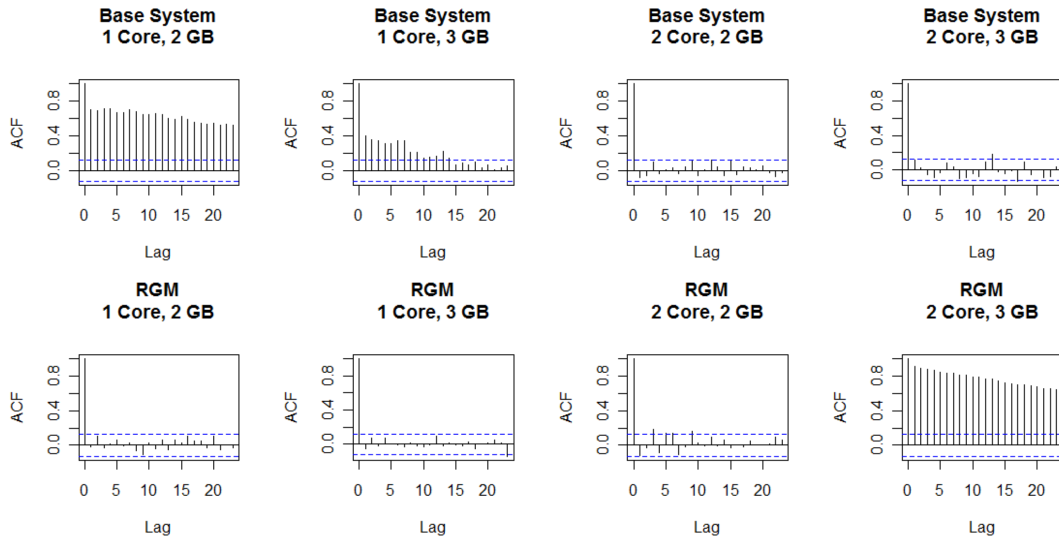


(b) Slide Show 2

Figure A.5: Autocorrelation Plots for PowerPoint



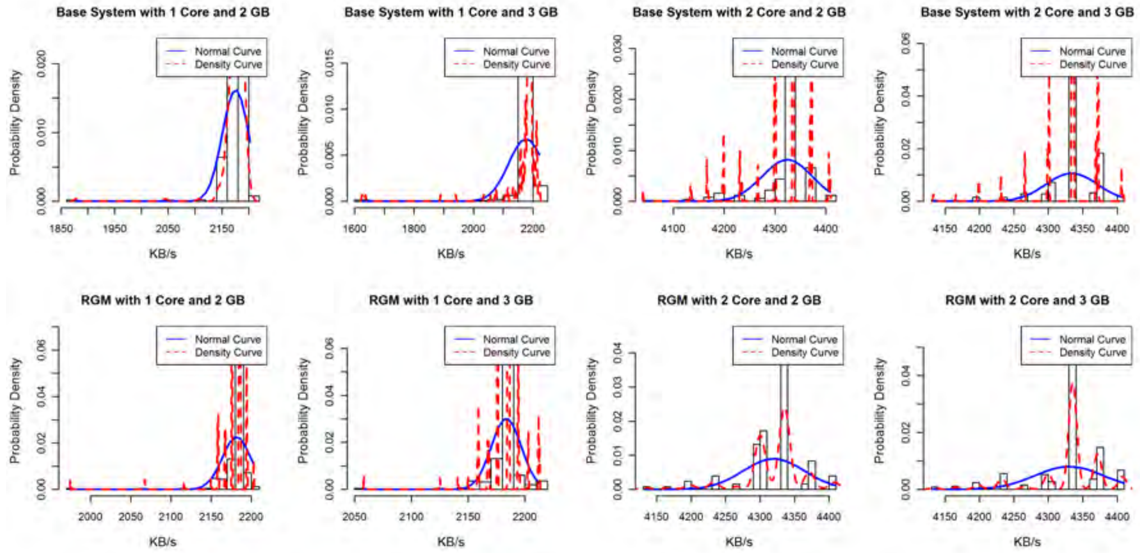
(a) Document 1



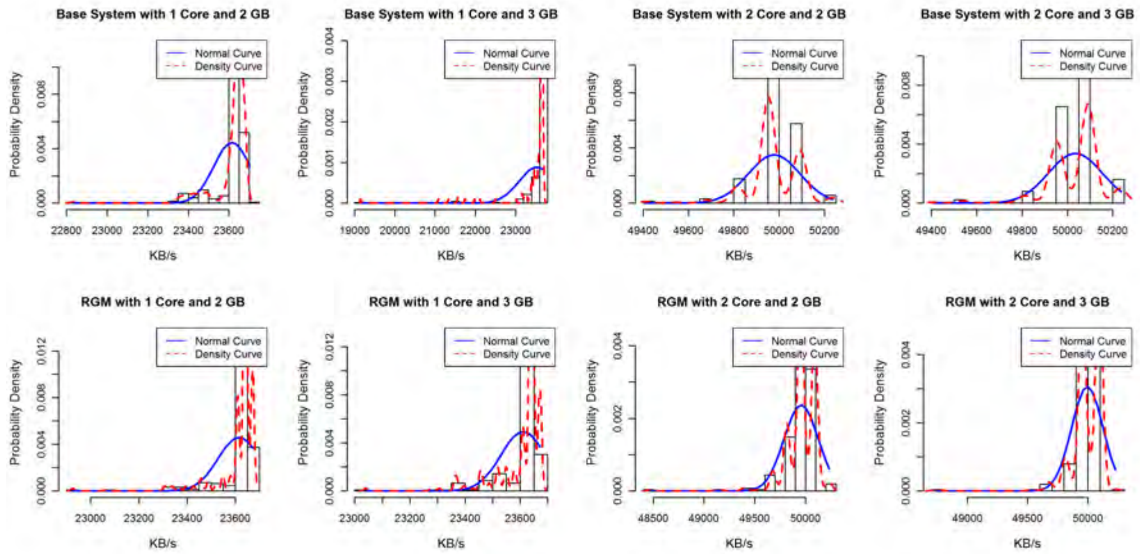
(b) Document 2

Figure A.6: Autocorrelation Plots for Word

Appendix B: Distribution Plots

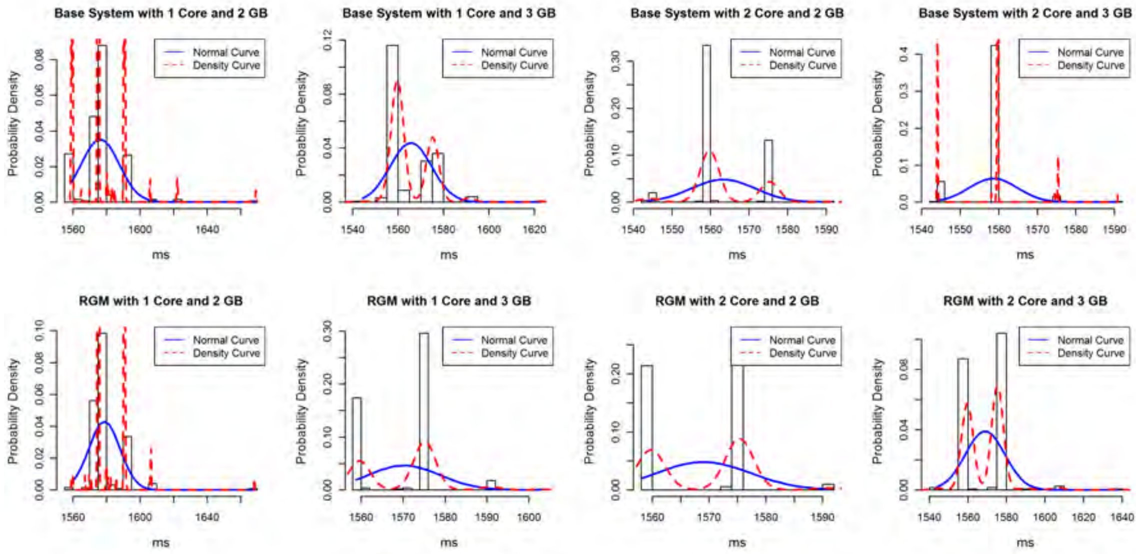


(a) Compression

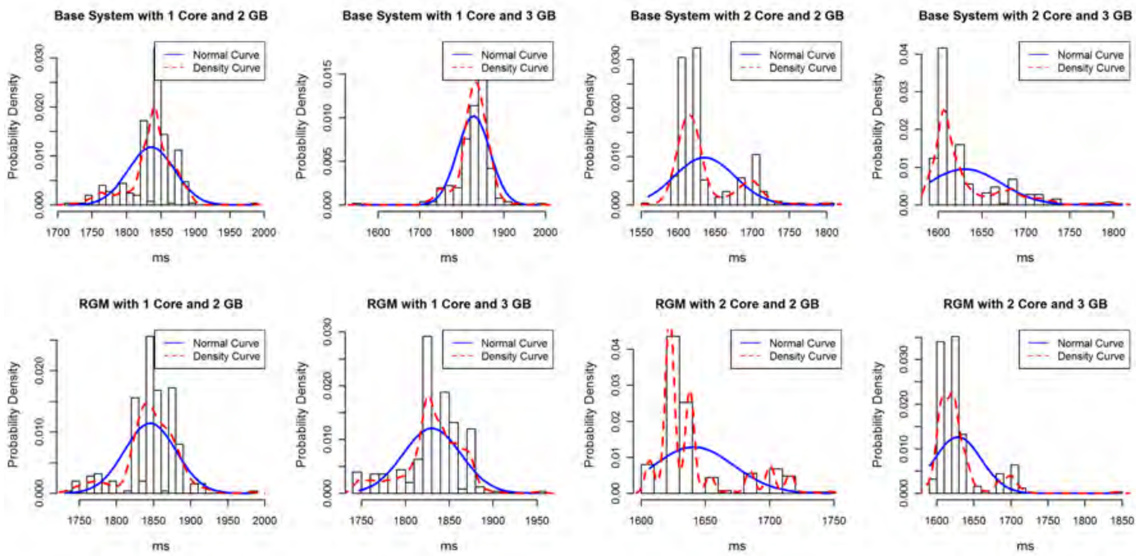


(b) Decompression

Figure B.1: Distribution Plot For 7-Zip

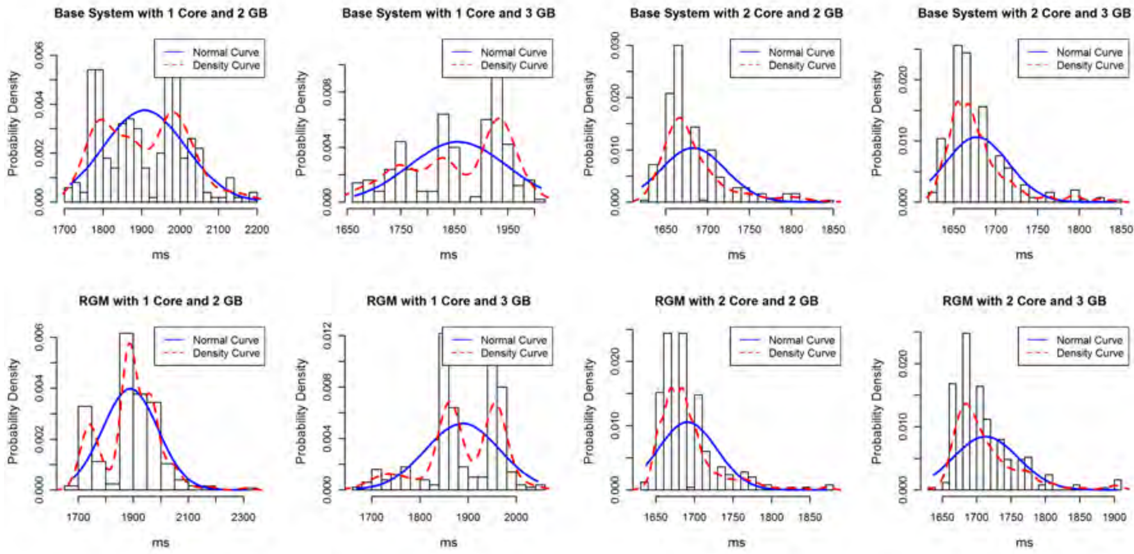


(a) PDF 1

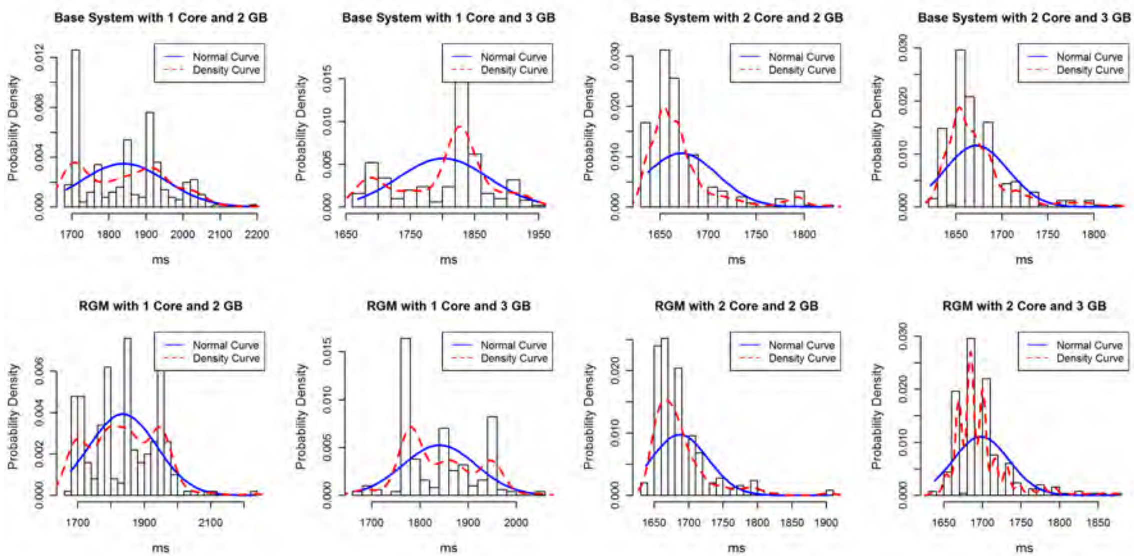


(b) PDF 2

Figure B.2: Distribution Plot For Acrobat Reader

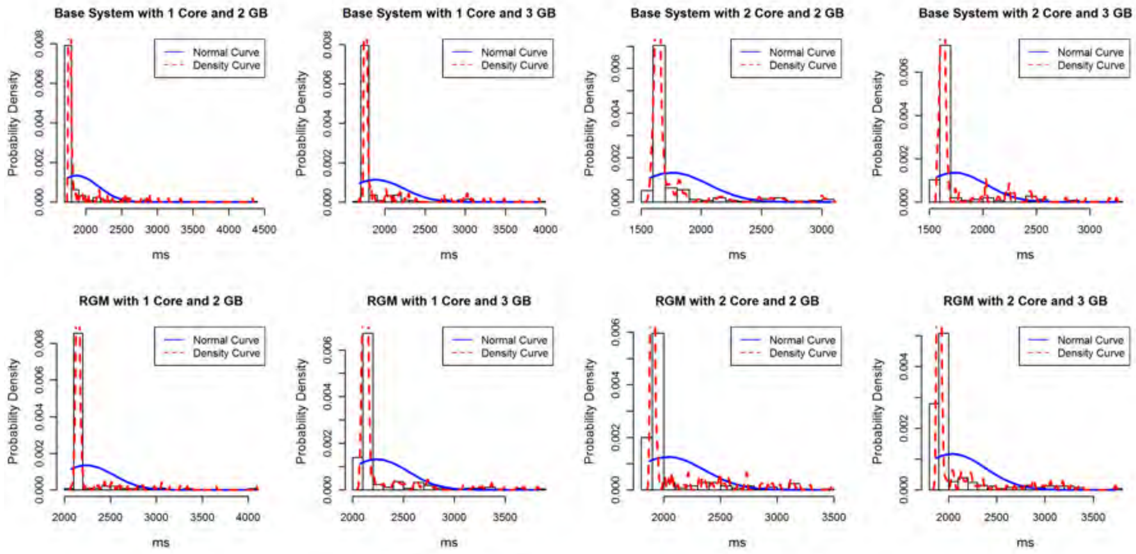


(a) Spreadsheet 1

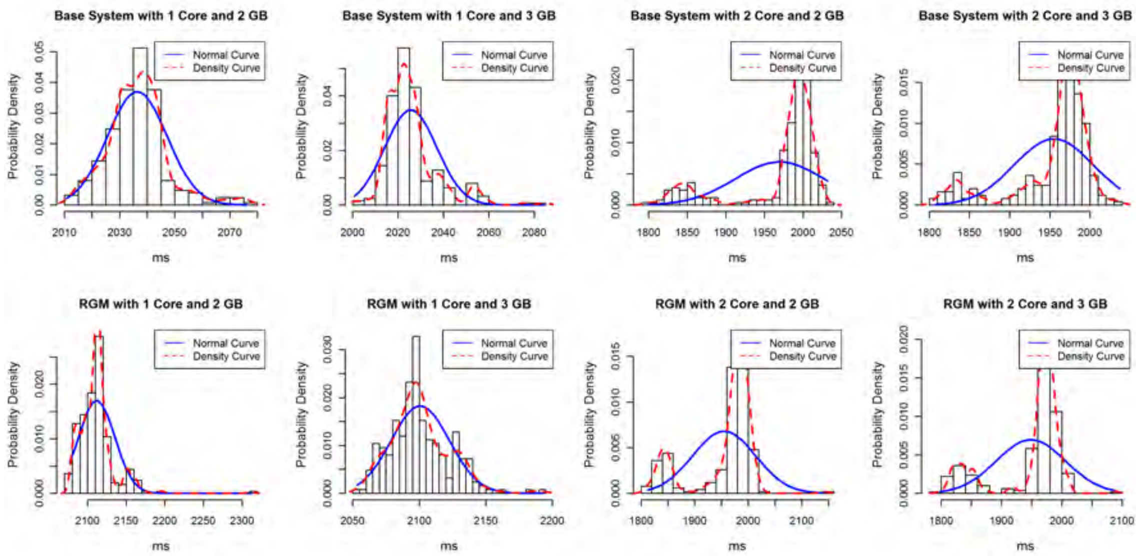


(b) Spreadsheet 2

Figure B.3: Distribution Plot For Excel

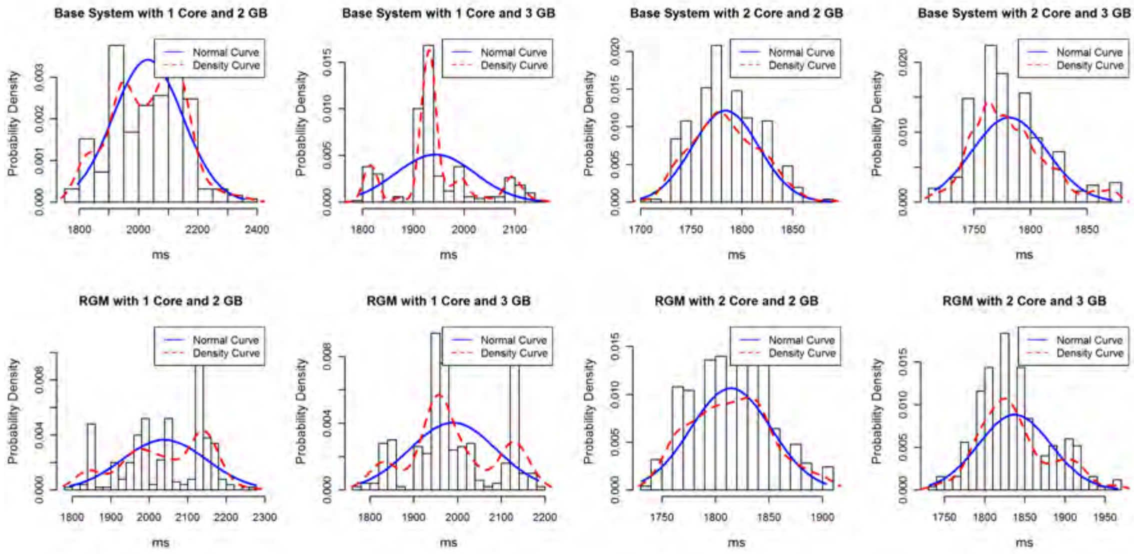


(a) Web Page 1

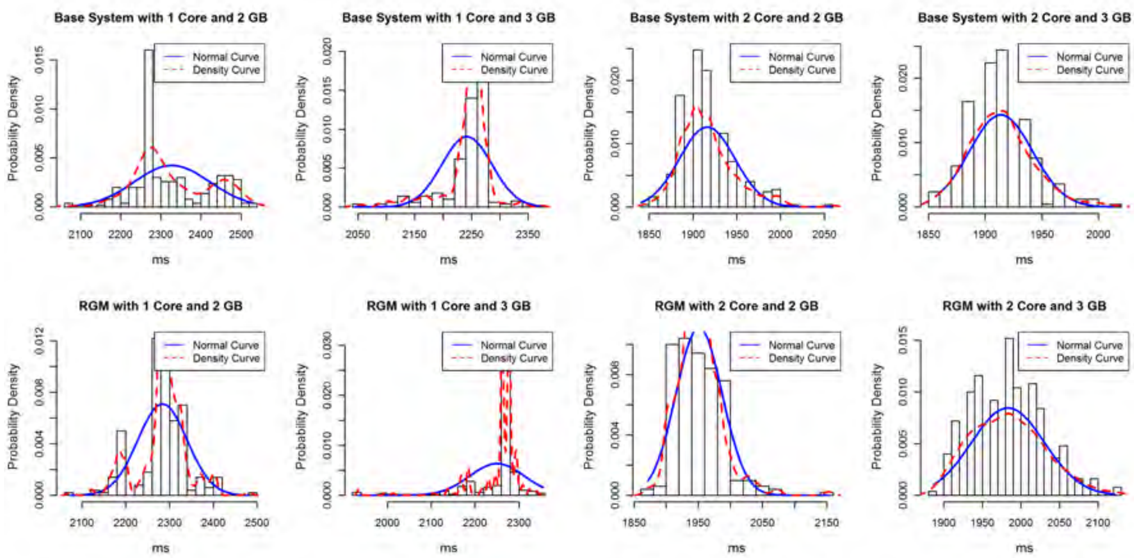


(b) Web Page 2

Figure B.4: Distribution Plot For Internet Explorer

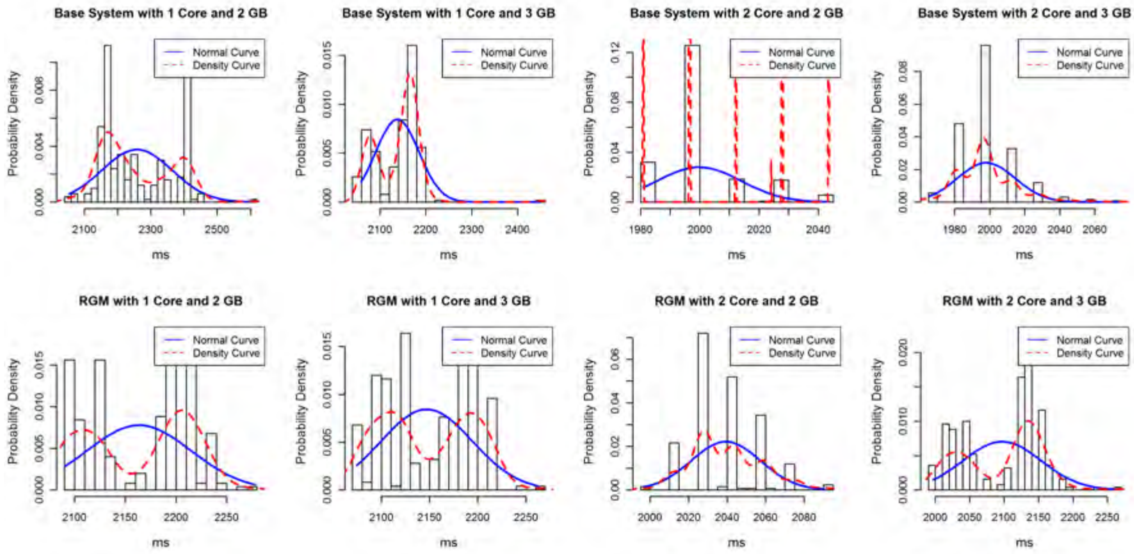


(a) Slide Show 1

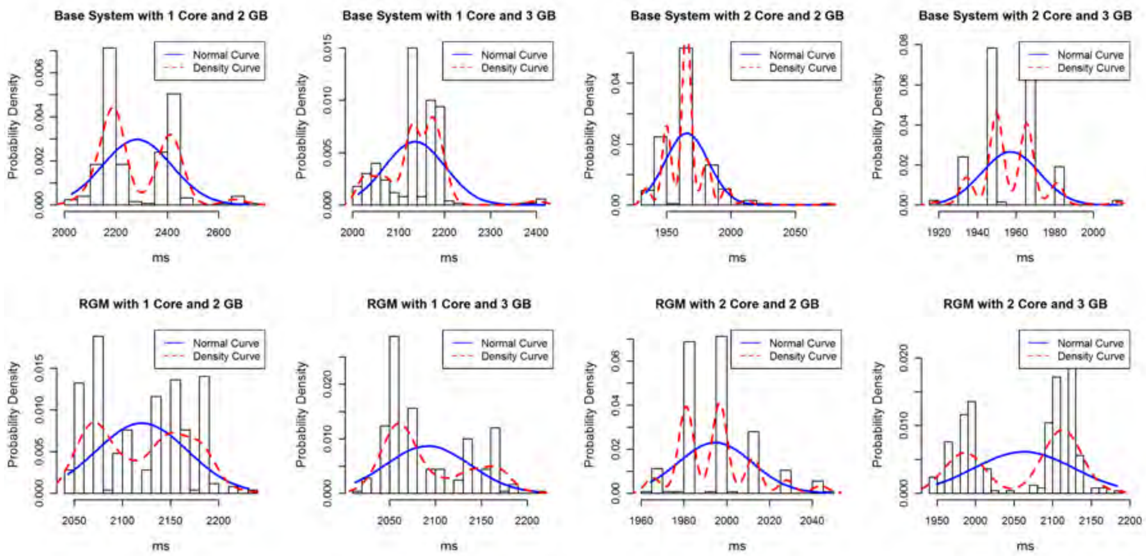


(b) Slide Show 2

Figure B.5: Distribution Plot For PowerPoint



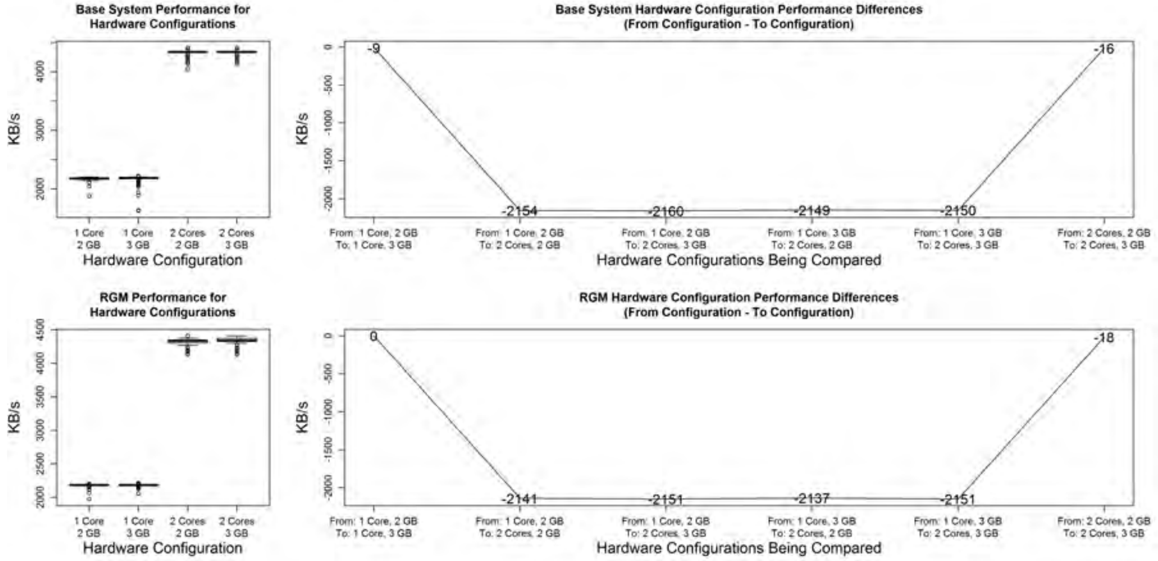
(a) Document 1



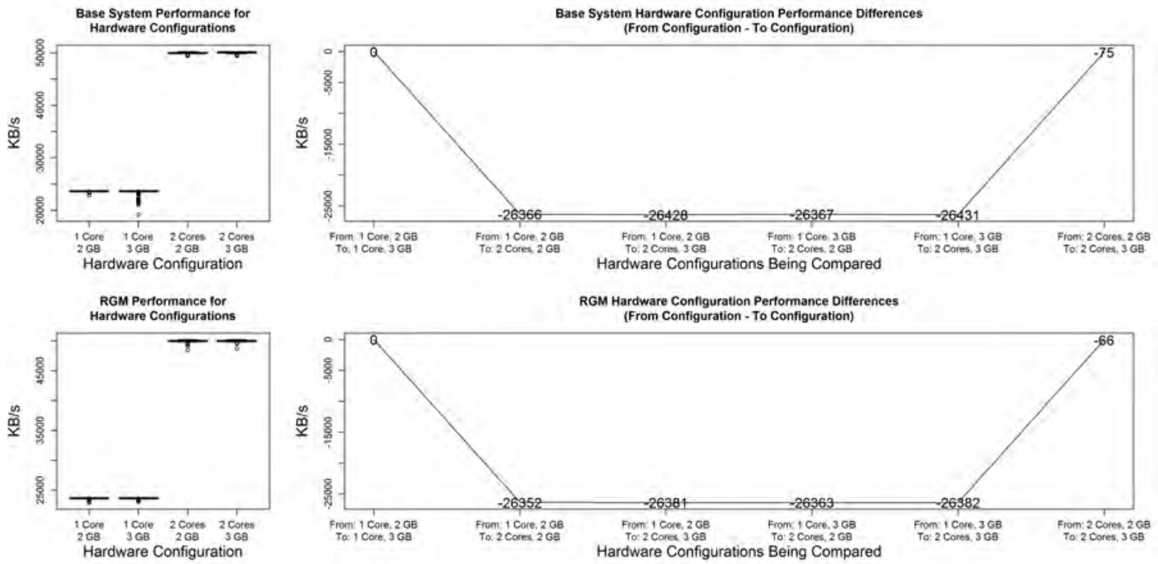
(b) Document 2

Figure B.6: Distribution Plot For Word

Appendix C: Effect of Hardware Configuration on Performance

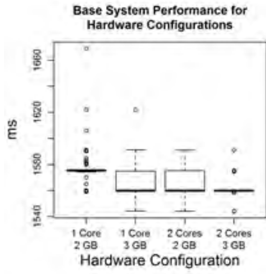


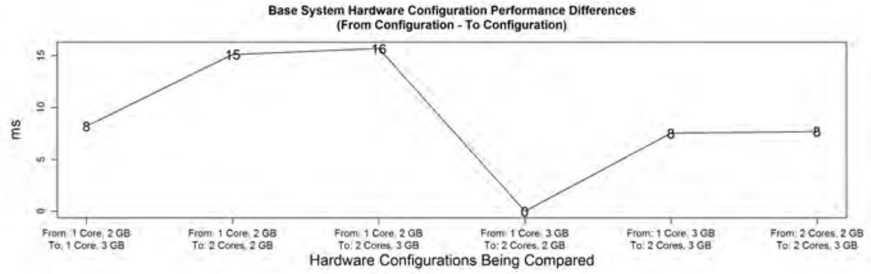
(a) Compression

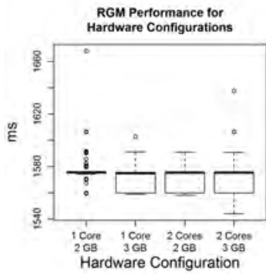


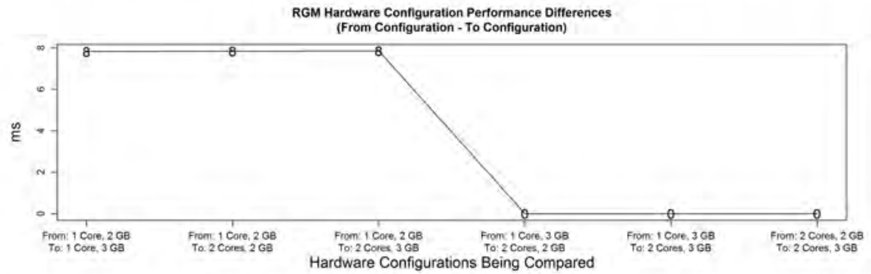
(b) Decompression

Figure C.1: Hardware Configuration Performance Results for 7-Zip

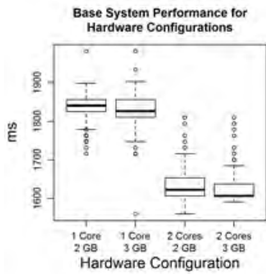


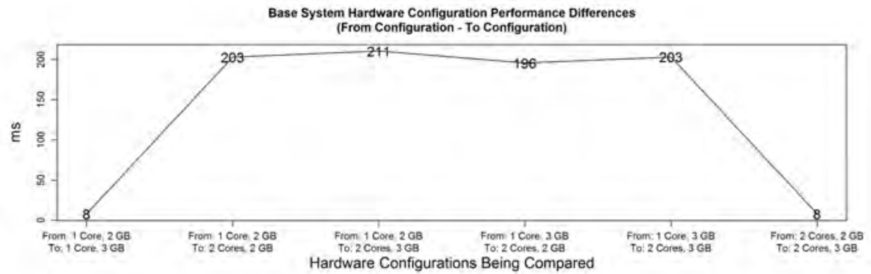


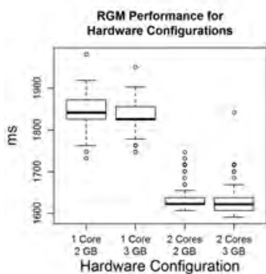


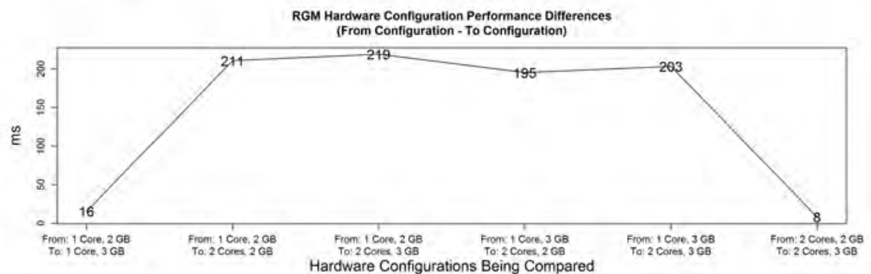


(a) PDF 1



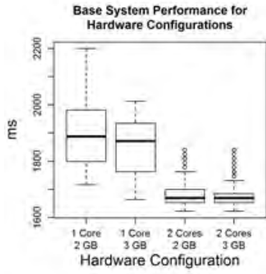


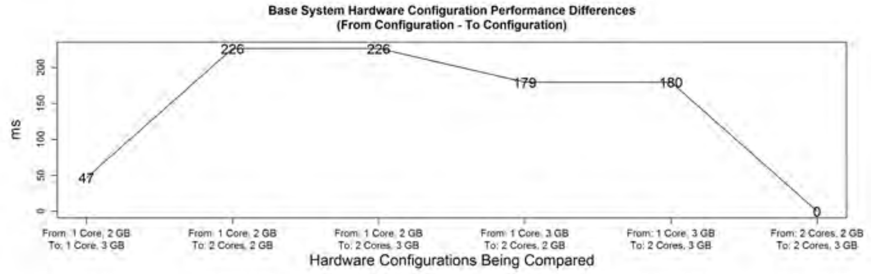


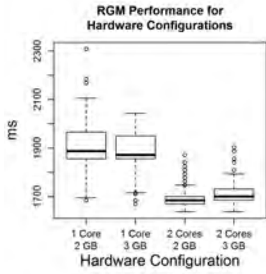


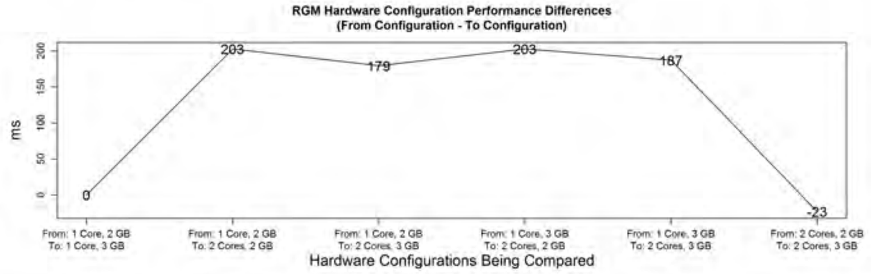
(b) PDF 2

Figure C.2: Hardware Configuration Performance Results for Acrobat Reader

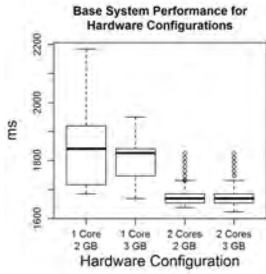


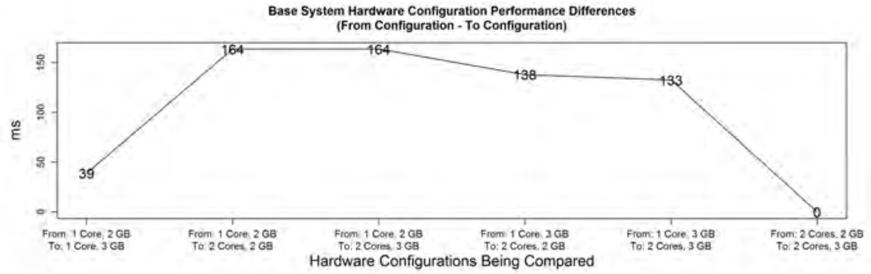


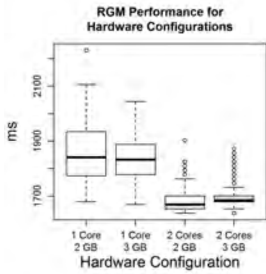


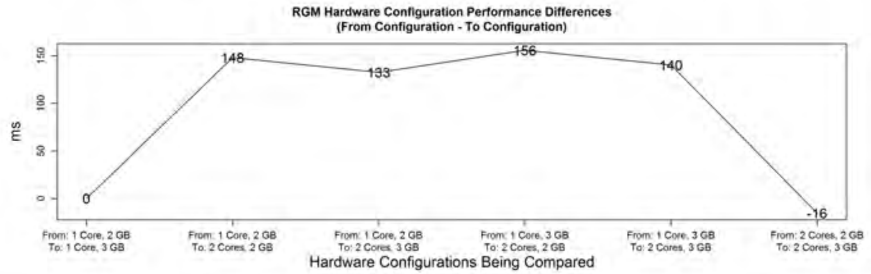


(a) Spreadsheet 1



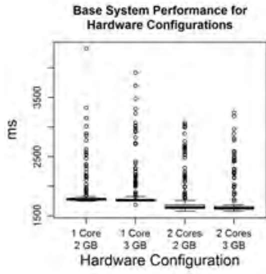


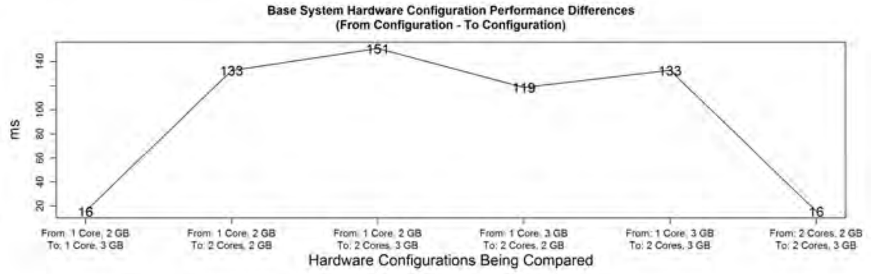


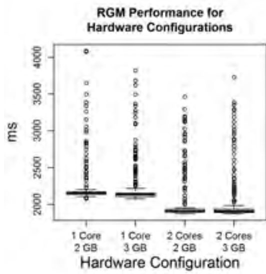


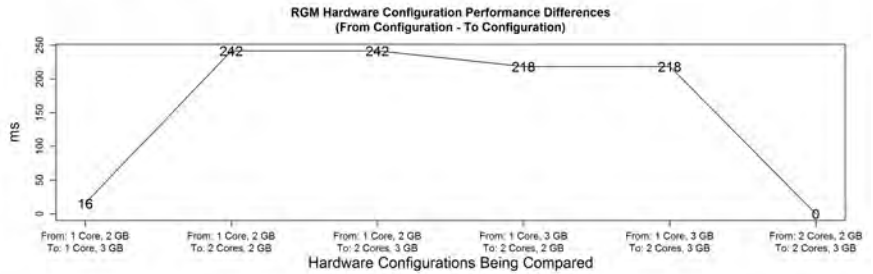
(b) Spreadsheet 2

Figure C.3: Hardware Configuration Performance Results for Excel

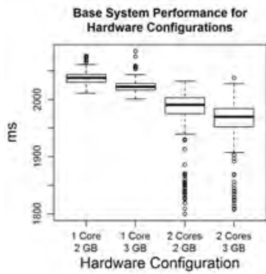


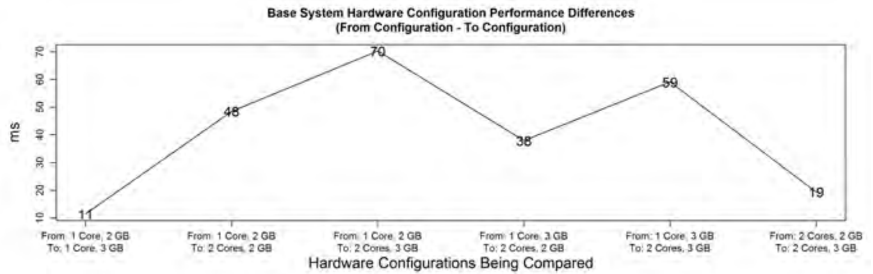


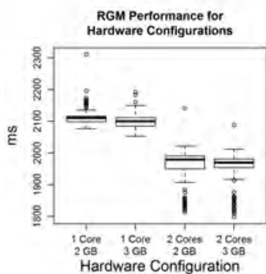


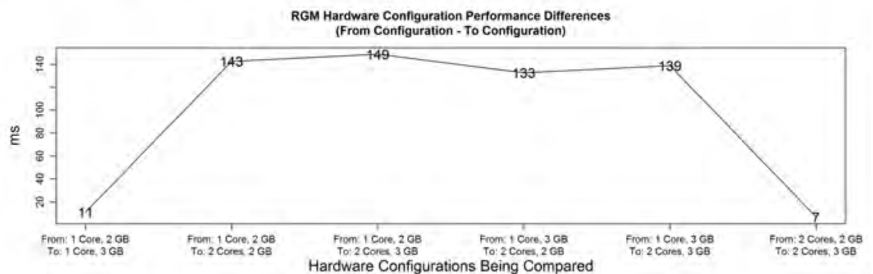


(a) Web Page 1



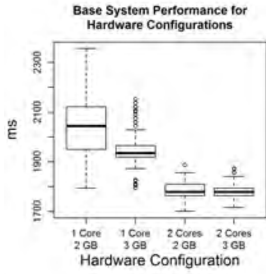


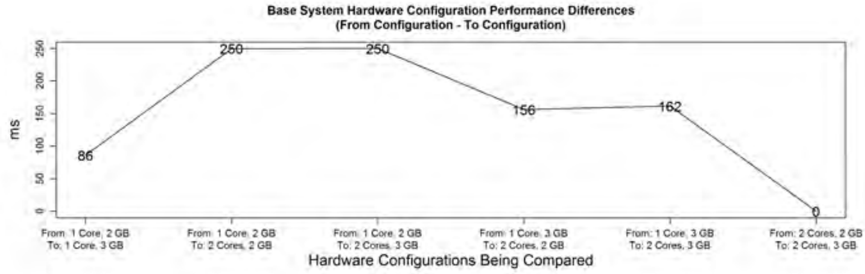


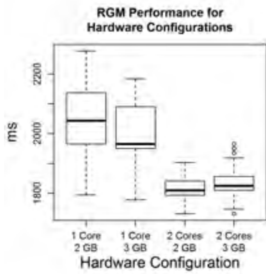


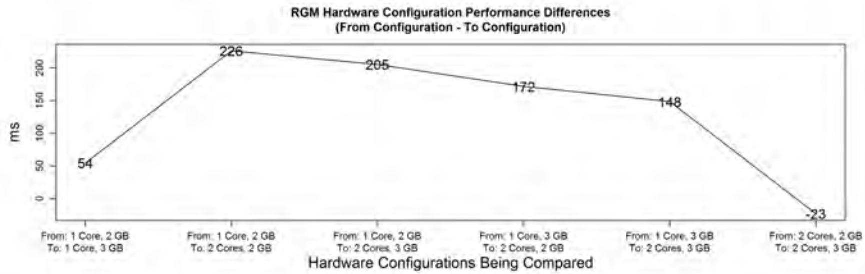
(b) Web Page 2

Figure C.4: Hardware Configuration Performance Results for Internet Explorer

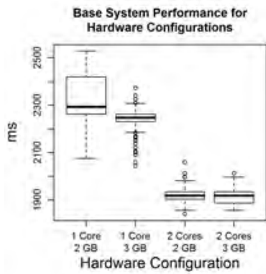


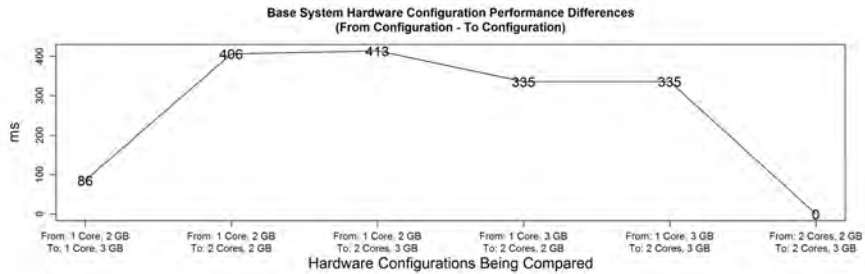


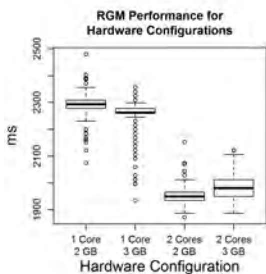


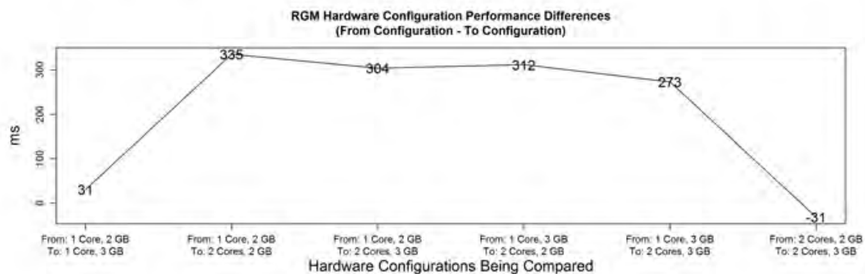


(a) Slide Show 1



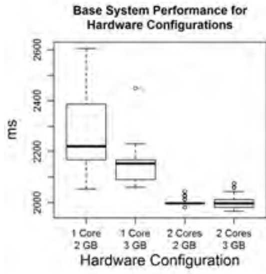


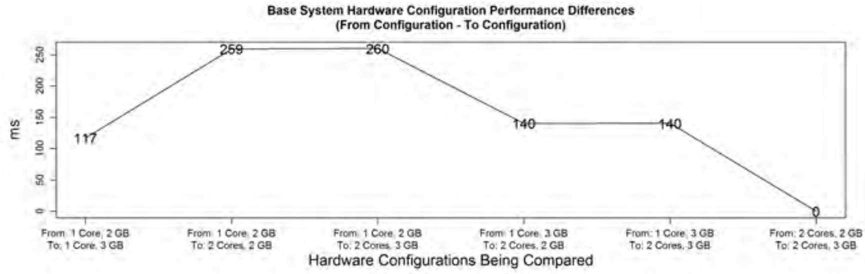


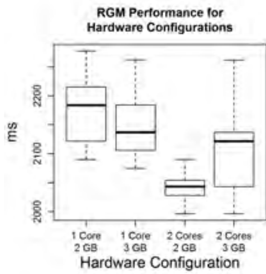


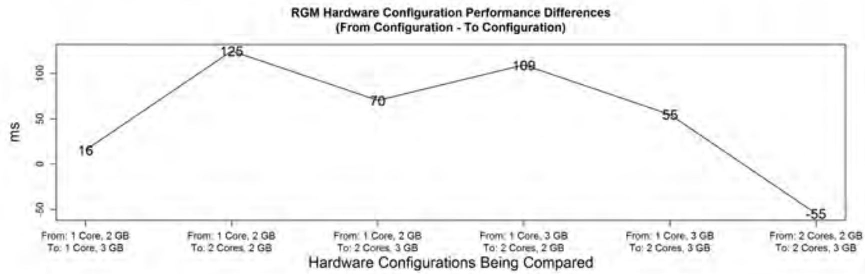
(b) Slide Show 2

Figure C.5: Hardware Configuration Performance Results for PowerPoint

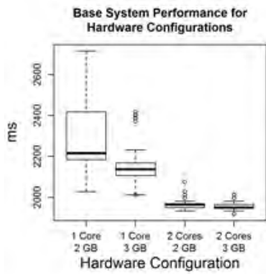


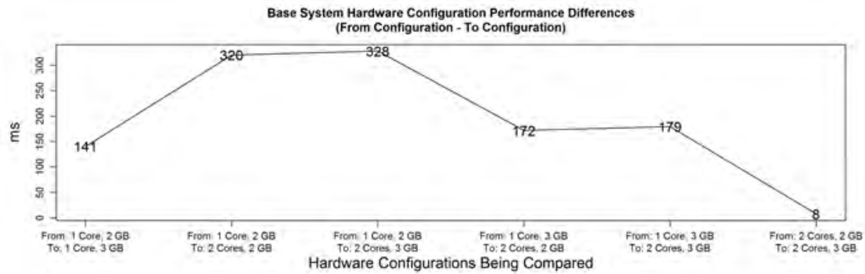


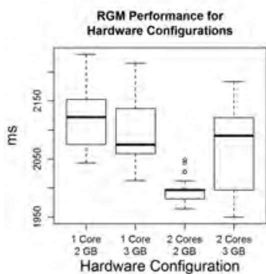


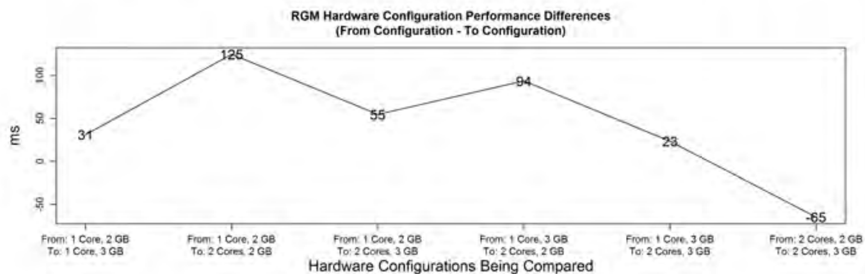


(a) Document 1









(b) Document 2

Figure C.6: Hardware Configuration Performance Results for Word

Appendix D: Performance Impact of Magnesium Object Manager Sandbox

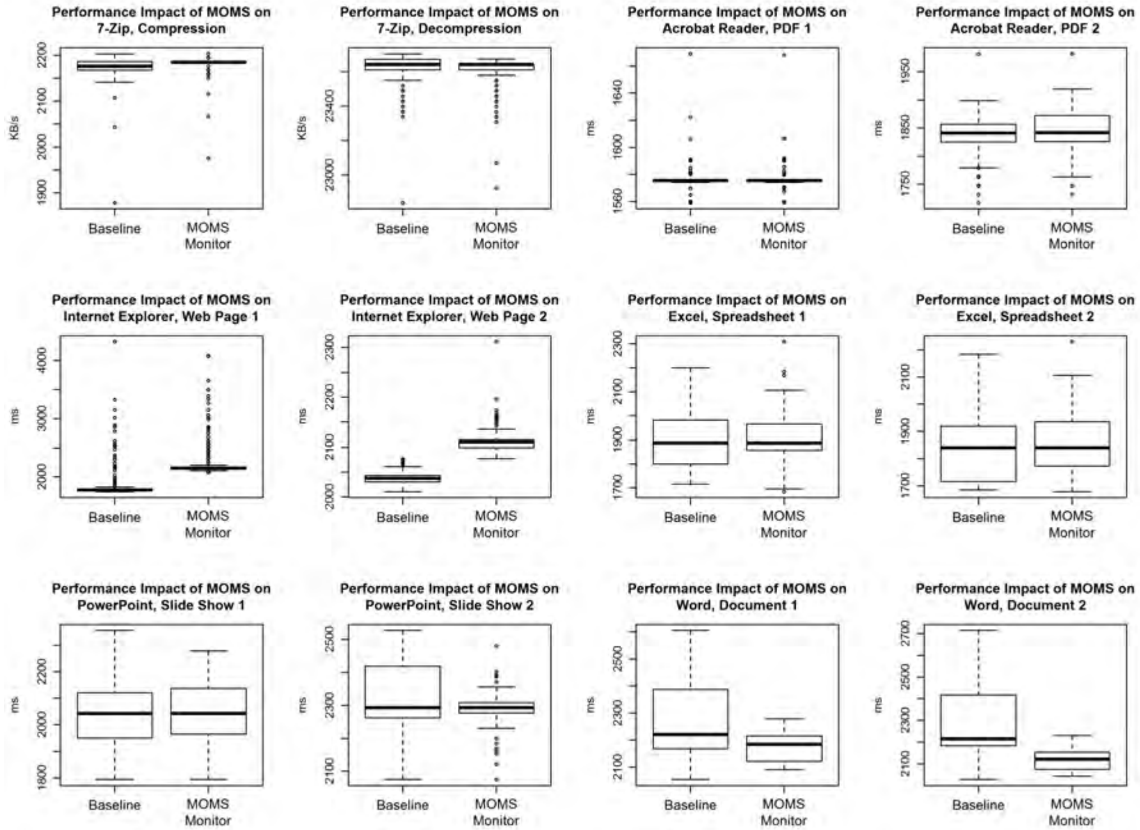


Figure D.1: Performance Impact of MOMS: 1 Core, 2 GB

Table D.1: Performance Impact Results of MOMS: 1 Core, 2 GB

Program	Document	Difference (Baseline - RGM)	Units	p-value
7-Zip	Compression	-1.000009	KB/s	1.259322e-06
	Decompression	1.598204e-05	KB/s	0.2136175
Acrobat Reader	PDF 1	-0.04093948	ms	0.05880068
	PDF 2	-10.81654	ms	4.651519e-05
Internet Explorer	Web Page 1	-374.3966	ms	4.084449e-57
	Web Page 2	-73.52848	ms	2.233854e-83
Excel	Spreadsheet 1	15.62357	ms	0.04867886
	Spreadsheet 2	0.004699188	ms	0.8958299
PowerPoint	Slide Show 1	-15.53777	ms	0.1704044
	Slide Show 2	16.28919	ms	0.002637716
Word	Document 1	77.96559	ms	4.604308e-20
	Document 2	125.0297	ms	8.180757e-55

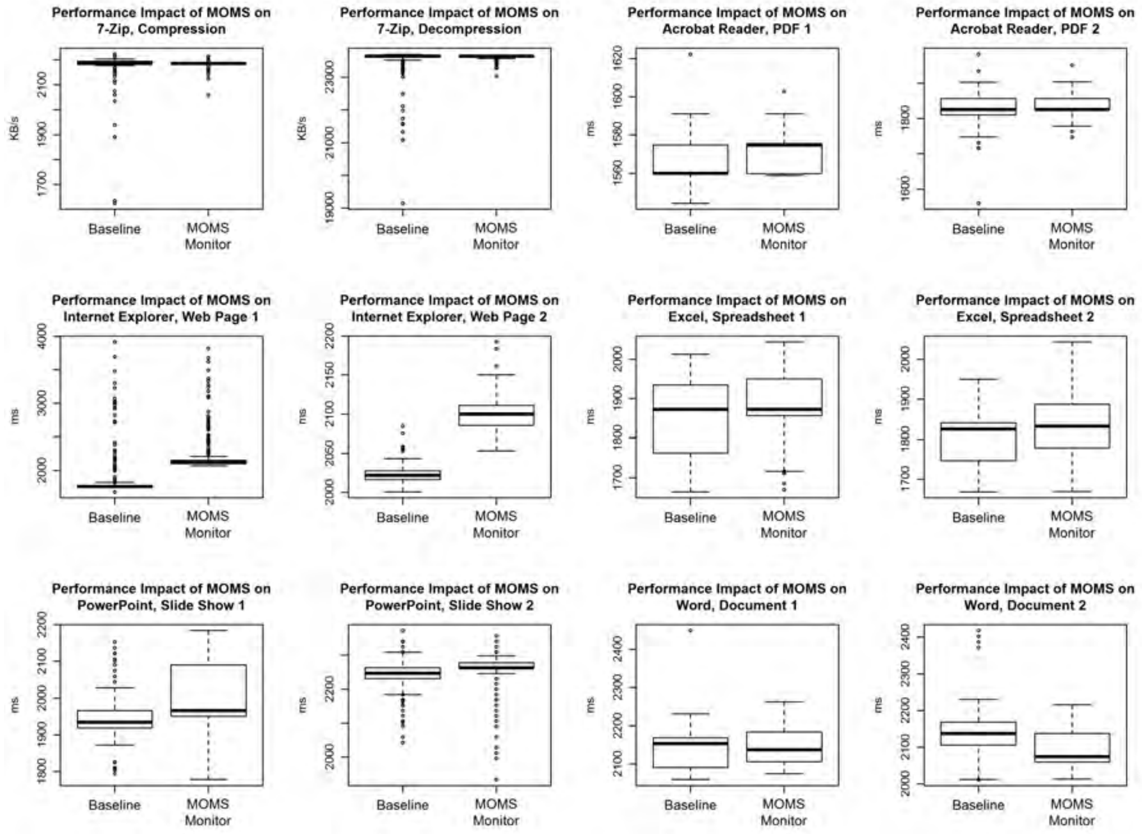


Figure D.2: Performance Impact of MOMS: 1 Core, 3 GB

Table D.2: Performance Impact Results of MOMS: 1 Core, 3 GB

Program	**Document**	**Difference (Baseline - RGM)**	**Units**	**p-value**
7-Zip	Compression	4.297132e-05	KB/s	0.007803446
	Decompression	4.981807e-05	KB/s	0.002834431
Acrobat Reader	PDF 1	-0.6517044	ms	1.389387e-07
	PDF 2	-0.03653842	ms	0.4229165
Internet Explorer	Web Page 1	-358.9382	ms	7.045827e-53
	Web Page 2	-73.61211	ms	7.865689e-83
Excel	Spreadsheet 1	-31.1707	ms	9.01863e-07
	Spreadsheet 2	-31.20813	ms	2.177921e-05
PowerPoint	Slide Show 1	-31.22941	ms	1.135776e-13
	Slide Show 2	-15.60341	ms	4.65167e-12
Word	Document 1	-15.59549	ms	0.0009502865
	Document 2	46.84255	ms	7.914954e-16

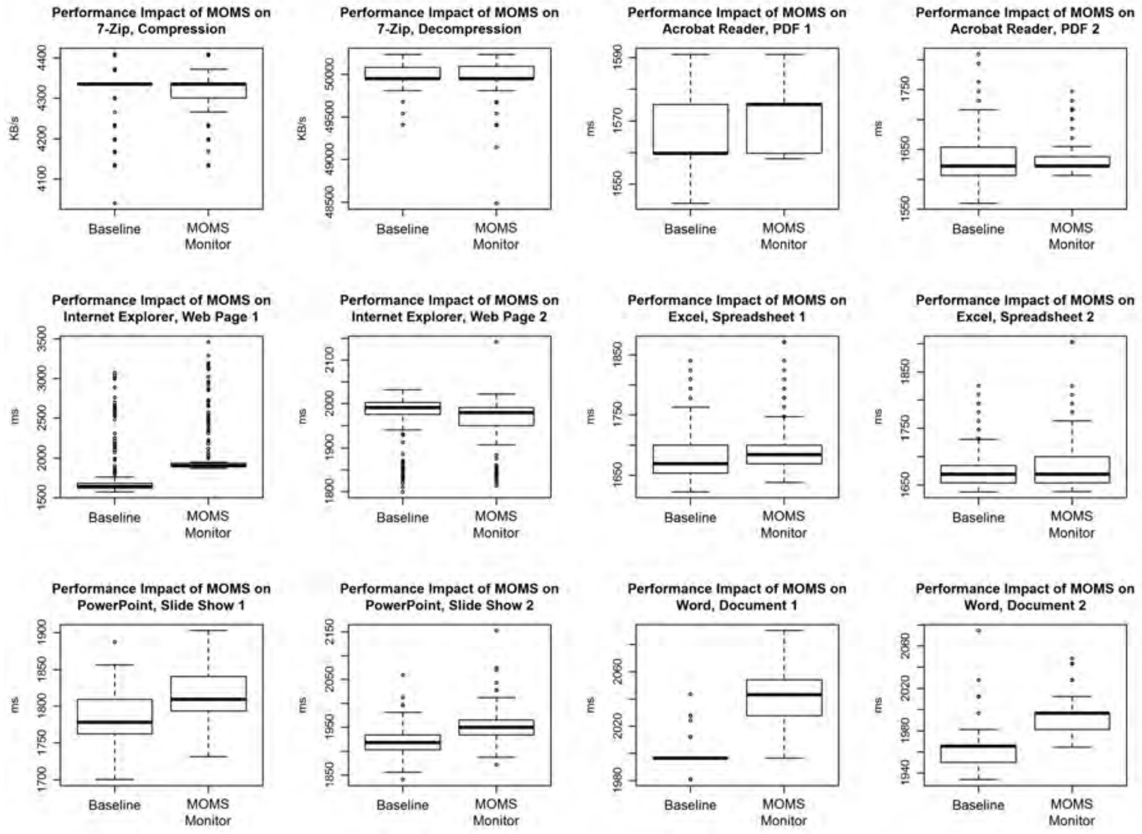


Figure D.3: Performance Impact of MOMS: 2 Cores, 2 GB

Table D.3: Performance Impact Results of MOMS: 2 Core, 2 GB

Program	**Document**	**Difference (Baseline - RGM)**	**Units**	**p-value**
7-Zip	Compression	5.13232e-06	KB/s	0.00500766
	Decompression	2.628481e-05	KB/s	0.6973014
Acrobat Reader	PDF 1	-0.1462755	ms	3.266672e-13
	PDF 2	-15.5056	ms	5.370356e-06
Internet Explorer	Web Page 1	-265.2194	ms	1.136964e-51
	Web Page 2	13.23786	ms	1.292535e-09
Excel	Spreadsheet 1	-15.465	ms	9.727734e-05
	Spreadsheet 2	-15.57145	ms	3.298751e-08
PowerPoint	Slide Show 1	-31.19289	ms	1.96343e-18
	Slide Show 2	-31.25074	ms	3.227453e-28
Word	Document 1	-41.81169	ms	1.305241e-66
	Document 2	-31.16406	ms	5.532732e-54

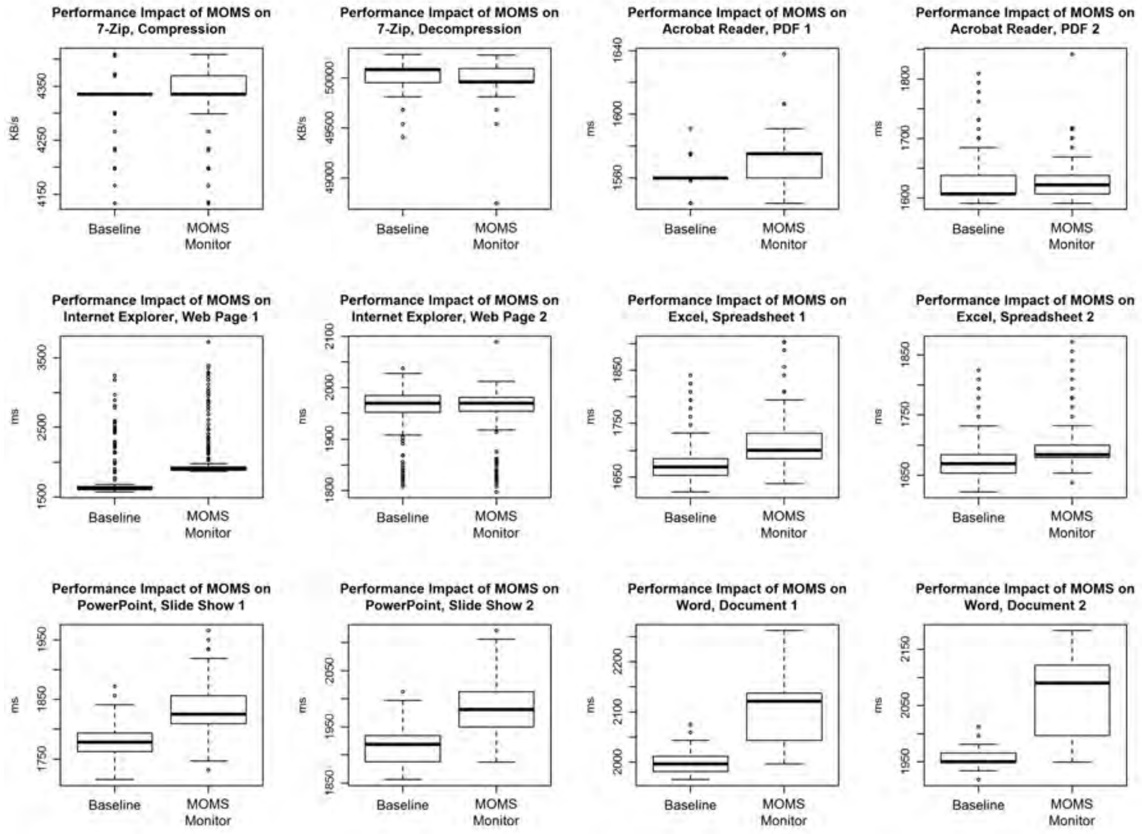


Figure D.4: Performance Impact of MOMS: 2 Cores, 3 GB

Table D.4: Performance Impact Results of MOMS: 2 Core, 3 GB

Program	**Document**	**Difference (Baseline - RGM)**	**Units**	**p-value**
7-Zip	Compression	3.125371e-05	KB/s	0.8182311
	Decompression	1.791934e-05	KB/s	0.0006328919
Acrobat Reader	PDF 1	-15.50572	ms	7.213207e-25
	PDF 2	-0.05368079	ms	0.0266545
Internet Explorer	Web Page 1	-280.7928	ms	1.180738e-47
	Web Page 2	0.8109125	ms	0.4801655
Excel	Spreadsheet 1	-31.18607	ms	1.793086e-25
	Spreadsheet 2	-31.12865	ms	1.538028e-22
PowerPoint	Slide Show 1	-46.94565	ms	3.390136e-42
	Slide Show 2	-62.45836	ms	1.033036e-53
Word	Document 1	-124.7062	ms	1.477834e-70
	Document 2	-140.3046	ms	7.622166e-68

Bibliography

[1] "About Atom Tables", September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/ms649053%28v=vs.85%29.aspx. Documentation.

[2] "About Synchronization", September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/ms681924%28v=vs.85%29.aspx.

[3] "About the Windows Driver Kit (WDK)", 2012. URL http://msdn.microsoft.com/en-us/windows/hardware/gg487428.

[4] "About Window Stations and Desktops", September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/ms681928%28v=vs.85%29.aspx.

[5] "Access Mask", September 2011. URL http://msdn.microsoft.com/en-us/library/windows/hardware/ff538834%28v=VS.85%29.aspx.

[6] "Access Rights for Access-Token Objects", November 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa374905%28v=vs.85%29.aspx.

[7] "AppArmor", January 2012. URL http://wiki.apparmor.net/index.php/Main_Page.

[8] "Attribute - $FILE_NAME (0x30)". URL http://inform.pucp.edu.pe/~inf232/Ntfs/ntfs_doc_v0.5/attributes/file_name.html.

[9] "avast! Pro Antivirus 6", 2012. URL http://www.avast.com/en-us/pro-antivirus.

[10] Bishop, Matt. *Computer Security: Art and Science*. Pearson Education, Inc., 2010.

[11] "Comodo Firewall", 2012. URL http://www.comodo.com/home/internet-security/firewall.php.

[12] "CreateNamedPipe function". Website, September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa365150%28v=vs.85%29.aspx.

[13] "CreateProcessNotifyEx routine", December 2011. URL http://msdn.microsoft.com/en-us/library/windows/hardware/ff542860%28v=vs.85%29.aspx.

[14] "Creating Symbolic Links", September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa363878%28v=vs.85%29.aspx.

[15] d0c_s4vage, sinn3r, and bannedit. "MS11-050 IE mshtml!CObjectElement Use After Free". Website, June 2011. URL http://www.metasploit.com/modules/exploit/windows/browser/ms11_050_mshtml_cobjectelement.

[16] Dodonov, Evgueni, Joelle Quaini Sousa, and Hélio Crestana Guardia. "GridBox: securing hosts from malicious and greedy applications". *Proceedings of the 2nd workshop on Middleware for grid computing*, MGC '04, 17–22. ACM, New York, NY, USA, 2004. ISBN 1-58113-950-0. URL http://doi.acm.org/10.1145/1028493.1028496.

[17] "Fsutil hardlink", April 2010. URL http://technet.microsoft.com/en-us/library/cc788097.aspx.

[18] "GetFileInformationByHandleEx function", September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa364953%28v=vs.85%29.aspx.

[19] Greamo, C. and A. Ghosh. "Sandboxing and Virtualization: Modern Tools for Combating Malware". *Security Privacy, IEEE*, 9(2):79 –82, march-april 2011. ISSN 1540-7993.

[20] "Hard Links and Junctions", September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa365006%28v=vs.85%29.aspx.

[21] "Introducing Sandboxie", 2012. URL http://www.sandboxie.com/.

[22] "I/O Completion Ports". Website, September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa365198%28v=vs.85%29.aspx.

[23] jduck. "Adobe CoolType SING Table "uniqueName" Stack Buffer Overflow". Website, September 2010. URL http://www.metasploit.com/modules/exploit/windows/browser/adobe_cooltype_sing.

[24] jduck. "Sun Java Runtime New Plugin docbase Buffer Overflow". Website, October 2010. URL http://www.metasploit.com/modules/exploit/windows/browser/java_docbase_bof.

[25] "Job Objects". Website, September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/ms684161%28v=vs.85%29.aspx.

[26] Kabacoff, Robert I. *R in Action: Data analysis and graphics with R*. Manning Publications Co., Shelter Island, NY, 2011.

[27] chung Lam, Lap and Tzi cker Chiueh. "Automatic extraction of accurate application-specific sandboxing policy". *Military Communications Conference, 2005. MILCOM 2005. IEEE*, 713 –719 Vol. 2. oct. 2005.

[28] Lanzi, Andrea, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. "AccessMiner: using system-centric models for malware protection". *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, 399–412. ACM, New York, NY, USA, 2010. ISBN 978-1-4503-0245-6. URL http://doi.acm.org/10.1145/1866307.1866353.

[29] Li, Wei, Lap chung Lam, and Tzi cker Chiueh. "How to Automatically and Accurately Sandbox Microsoft IIS". *Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual*, 213 –222. dec. 2006. ISSN 1063-9527.

[30] Marinescu, Adrian and The Channel 9 Team. "Windows, NT Object Manager", June 2005. URL http://channel9.msdn.com/Shows/Going+Deep/Windows-NT-Object-Manager.

[31] natron. "Internet Explorer Unsafe Scripting Misconfiguration". Website, September 2010. URL http://www.metasploit.com/modules/exploit/windows/browser/ie_unsafe_scripting.

[32] "NTFS File Types", 2010. URL http://www.ntfs.com/ntfs-files-types.htm.

[33] "Object Directories". Website, December 2011. URL http://msdn.microsoft.com/en-us/library/windows/hardware/ff557755%28v=vs.85%29.aspx.

[34] "Object Namespaces". Windows Dev Center, September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/ms684295%28v=vs.85%29.aspx.

[35] "OB_PRE_CREATE_HANDLE_INFORMATION structure". Website, December 2011. URL http://msdn.microsoft.com/en-us/library/windows/hardware/ff558725%28v=vs.85%29.aspx.

[36] "Parts of the Access Control Model", November 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa374862%28v=vs.85%29.aspx.

[37] passerby, d0c_s4vage, and jduck. "Internet Explorer CSS Recursive Import Use After Free". Website, November 2010. URL http://www.metasploit.com/modules/exploit/windows/browser/ms11_003_ie_css_import.

[38] Radhakrishnan, M. and J.A. Solworth. "Quarantining Untrusted Entities: Dynamic Sandboxing Using LEAP". *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, 211 –220. dec. 2007. ISSN 1063-9527.

[39] "Registry Key Security and Access Rights". Website, September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/ms724878%28v=vs.85%29.aspx.

[40] Riondato, Matteo. "Jails". FreeBSD. URL http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html.

[41] Russinovich, Mark E., David A. Solomon, and Alex Ionescu. *Windows Internals: Fifth Edition*. Microsoft Press, 2009.

[42] "SACL Access Right". Website, November 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa379321%28v=vs.85%29.aspx.

[43] Schmid, M., F. Hill, and A.K. Ghosh. "Protecting data from malicious software". *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, 199 – 208. 2002. ISBN 0-7695-1828-1. ISSN 1063-9527.

[44] "Section Objects and Views". Website, December 2011. URL http://msdn.microsoft.com/en-us/library/windows/hardware/ff563684%28v=vs.85%29.aspx.

[45] "Security-Enhanced Linux", January 2009. URL http://www.nsa.gov/research/selinux/index.shtml.

[46] Stevens, Didier. "Escape From PDF". Website, March 2010. URL http://www.exploit-db.com/exploits/11987/.

[47] "Synchronization Object Security and Access Rights". Website, September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/ms686670%28v=vs.85%29.aspx.

[48] "Sysinternals File and Disk Utilities". URL http://technet.microsoft.com/en-us/sysinternals/bb545046.

[49] Tan, Lin, E.M. Chan, R. Farivar, N. Mallick, J.C. Carlyle, F.M. David, and R.H. Campbell. "iKernel: Isolating Buggy and Malicious Device Drivers Using Hardware Virtualization Support". *Dependable, Autonomic and Secure Computing, 2007. DASC 2007. Third IEEE International Symposium on*, 134 –144. sept. 2007.

[50] "Thread Security and Access Rights". Website, September 2011. URL http://msdn.microsoft.com/en-us/library/windows/desktop/ms686769%28v=vs.85%29.aspx.

[51] Weese, Bill. "About the ACCESS_MASK Structure", April 2010. URL http://blogs.msdn.com/b/openspecification/archive/2010/04/01/about-the-access-mask-structure.aspx.

[52] "What's in a (Process) Name? Obtaining A Useful Name for the Executable Image in a Process". *The NT Insider*, 13(4), September 2006. URL http://www.osronline.com/article.cfm?article=472.

[53] "Windows 7 Object Headers", January 2011. URL http://www.codemachine.com/article_objectheader.html.

[54] "ZwCreateFile routine". Website, December 2011. URL http://msdn.microsoft.com/en-us/library/windows/hardware/ff566424%28v=vs.85%29.aspx.

[55] "ZwCreateSection routine". Website, December 2011. URL http://msdn.microsoft.com/en-us/library/windows/hardware/ff566428%28v=vs.85%29.aspx.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 074-0188
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**			
1. REPORT DATE *(DD-MM-YYYY)* 22-03-2012	**2. REPORT TYPE** Master's Thesis		**3. DATES COVERED** *(From – To)* Sep 2010 – Mar 2012
4. TITLE AND SUBTITLE Magnesium Object Manager Sandbox, A More Effective Sandbox Method for Windows 7			**5a. CONTRACT NUMBER**
			5b. GRANT NUMBER
			5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Gilligan, Martin, A., 2d Lt, USAF			**5d. PROJECT NUMBER**
			5e. TASK NUMBER
			5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology, Graduate School of Engineering and Management (AFIT/EN), 2950 Hobson Way, WPAFB, OH 45433-7765			**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCE/ENG/12-05
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally Left Blank			**10. SPONSOR/MONITOR'S ACRONYM(S)**
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
A major issue in computer security is limiting the affects a program can have on a computer. One way is to place the program into a sandbox, a limited environment. Many attempts have been made to create a sandbox that maintains the usability of a program and effectively limits the effects of the program. Sandboxes that limit the resources programs can access, have succeeded. To test the effectiveness of a sandbox that limits the resources a program can access on Windows 7, the Magnesium Object Manager Sandbox (MOMS) is created. MOMS uses a kernel mode Windows component to monitor and limit the access rights to every resource.

Based on the performance data of a set of test programs, running with and without MOMS, and with different hardware configurations, the hardware configuration and MOMS has an impact to performance a normal user probably will not notice. For the exploits run against two of the test programs, none of the associated payloads successfully ran. While these tests are promising, they are limited in scope and further testing is required to increase their scope. Furthermore, based on analysis of MOMS, vulnerabilities exist, but they are straightforward to fix with further development.

15. SUBJECT TERMS
Windows, object manager, sandbox, performance comparison, function callbacks

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Lt Col Jeffrey W. Humphries (ENG)
REPORT	**ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	104	(937) 255-3636, x7253 Jeffrey.Humphries@afit.edu

Standard Form 298 (Rev: 8-98)
Prescribed by ANSI Std. Z39-18