Stony Brook University

Win32 Application Binary Streaming

A Thesis Presented

by

Aravind Akella

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

May 2010

Stony Brook University

The Graduate School

Aravind Akella

We, the thesis committee for the above candidate for the

Master of Science degree, hereby recommend

acceptance of this thesis.

Professor Tzi-cker Chiueh - Thesis Advisor
Department of Computer Science

Professor Scott Stoller – Chairperson of Defense
Department of Computer Science

Assistant Professor Jie Gao
Department of Computer Science

This thesis is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Win32 Application Binary Streaming

by

Aravind Akella

Master of Science

in

Computer Science
Stony Brook University
2010

The goal of this project is to centralize the patching and installation of applications. Given a windows server where all the applications like Microsoft Office are installed, a client should be able to open the shared binaries (executables and dynamically linked libraries (dlls)) installed on the server and run the application directly on the client without having to install it. The executable should actually run on the client (not a thin client scenario where the actual application is hosted on the server and the display is exported to the client).This model has the advantage of utilizing the computing power of the client. To achieve this objective, we intercept and redirect the windows system calls (using the Feather Weight Virtual Machine(FVM) framework) and user level APIs for COM. Whenever an application tries to open a particular dll we redirect it back to the shared location on the server which contains the actual dll. A local cache of dlls is maintained for faster access. Every time a dll is accessed from the cache, we check the server for a newer version of the dll (some update to the application might have changed the dll). The newer version is copied over to the cache and it erases the previous copy of the dll. When the server is down the application can run in disconnected mode using only the dlls from the local cache. User level APIs for COM are intercepted to enable Inprocess and Independent process COM. Finally porting of FVM framework to Vista SP1,SP2 and windows 7 is described.

To

My Family, Teachers and Friends

Table of Contents

List of Tables

List of Figures

Acknowledgements

First of all, I wish to sincerely thank Prof. Tzi-cker Chiueh for his guidance and support all through the project. I would also like to thank all the present and past colleagues at Rether Networks Inc (RNI). In particular, Pi-yuan Cheng for his insightful advice and help throughout my thesis work and Sheng-I Doong, President of Rether Networks, for her kind support.

Chapter 1

1 Introduction

Application deployment on end-user machines can be done in three major ways 1) Binary Streaming 2) Thin Client 3) Local Installation.

Binary Streaming application binaries and configurations are bundled into self-contained packages, which are stored and maintained centrally. An end user machine running a compatible OS fetches these packages from the server and runs them directly without local installation. Some of the packages may be cached locally for faster access.

In thin-client model, applications are installed on a centralized server. Only the display is exported to the client. The application maintenance costs are considerably lowered due to centralized binary installation and execution. This architecture, however, incurs large application runtime latency and higher network traffic load. This does not utilize the computation power of the client and treats them as dumb terminals. The server has to be a high-end machine to support multiple clients simultaneously.

In the local-installation model, the application is installed on very machine in the network. A network wide installation tool may be required to automate the installation process. Application management is difficult because all the machines in network have to be patched.

Binary Streaming combines the best of the above two models. It has the following advantages.[16]

- Centralized application management, with local execution.
- Accelerated application deployment.
- Ability to continue to use applications when off-line (in contrast to pure web-applications).
- Delivers fully featured desktop applications (in contrast to browser-driven web-applications)
- Simplified operating system migrations.

- Given the complexity of modern applications, many functions are never or seldom used, and pulling the application on demand is more efficient in terms of server, client and network usage.

In unix environment, binary streaming works by simply copying over the executable and the dependent libraries to the client machine on demand. But in Windows, applications can run only on the machines on which they are installed. This is mostly due to the registry settings, COM components and system services which cannot be easily replicated on other machines.
So a sophisticated mechanism is essential to get binary streaming working on Windows.

Whenever an application starts running it looks for its operating environment which consists of a) executable b) Dynamic linked libraries c) Configuration files d) COM components e) System services. To run this application on a client machine in Binary streaming mode, we need to reproduce the same execution environment that is available on an actual machine where the application is installed. We use the Feather Weight Virtual machine framework for this. We create a light weight virtual machine for the application on the client and emulate the server's execution environment.

Feather Weight Virtual machine (FVM) is a Windows based virtualization technique. The virtualization layer, in FVM, virtualizes the namespace by renaming the system resources, which forms the key idea of FVM. This renaming takes place at the OS system call interface. [1,2]

For Eg: If a process running in VM1 opens a file, "C:\foo\bar" it is redirected to "C:\VM1\foo\bar".

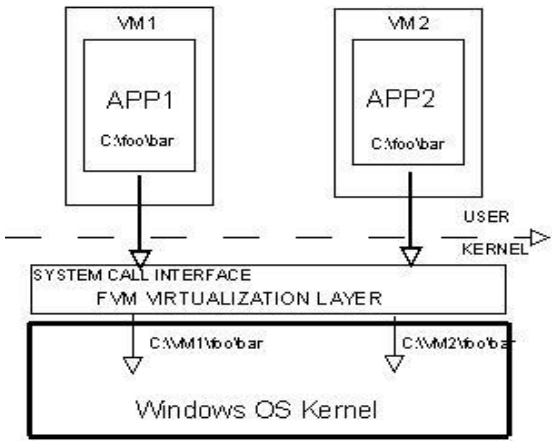

Figure 1: FVM Virtualization layer

We rely on Detours library for user level interceptions in FVM. Detours library replaces the first few instructions of the target API function with an unconditional jump to the user provided function, called a detour function. The replaced instructions from the target function are saved in a trampoline function, with an unconditional jump to the rest of code of the target function. Thus, a process s call to a target API is routed to the detour function, which can call its *trampoline function* when it requests service from the original target API function. We inject our custom DLL to the target process s address space. We use Detours express (32 bit) version for this.[4]

Binary Streaming uses a subset of the FVM framework. Not all system calls need to be intercepted by the Binary Streaming framework. In the following sections we describe the Binary Streaming framework in detail

Chapter 2

2 Related Work

Linux-VServer[5] is an OS-level virtualization mechanism for Linux. It is virtual private server implementation done by adding operating system-level virtualization capabilities to the Linux kernel. It is developed and distributed as open source software. Linux-VServer is a jail mechanism in that it can be used to securely partition resources on a computer system (such as the file system, CPU time, network addresses and memory) in such a way that processes cannot mount a denial-of-service attack on anything outside their partition.

VMware's ThinApp [6] is an application level virtualization technique. This technology allows Windows application to be processed into an executable file that can see its own version of Windows registry, file system and DLL"s. The virtual environment presented to the client is a merged view of the underlying physical and virtual resources, thereby allowing the virtualization layer to fool the application into thinking that it is running as if it were fully installed. ThinApp does not have any pre-installed components and does not require the installation of device drivers allowing applications to run from USB keys or network shares without ever requiring Administrator rights.

In thin client computing architecture, such as Microsoft Terminal Service [3] , application binaries are installed, maintained and executed on the server s CPU. An end user machine displays the result of application executions and replays user inputs via a special protocol such as Remote Desktop Protocol (RDP). X windows system [7] is a software that provides GUI to for networked systems. X was primarily designed to implement thin clients, where many people share the processing power of a time-sharing computer, and each person uses a networked terminal that has limited capability to draw the screen and accept user input.

In SaaS (Software as a service) model, the application is deployed over the internet. SaaS is generally associated by software professionals and business associates with business software and is typically thought of as a low-cost way for businesses to obtain rights to use software as needed versus licensing all devices with all applications. On-demand licensing enables the benefits of commercially licensed use without the associated complexity and potential high initial cost of equipping every device with the applications that are only used when needed. The application is managed centrally rather than at each customer site.[8]

The *shared binary service* architecture, which is widely used in the UNIX world, is similar to application streaming except that it does not require explicit application packaging, and it allows resource sharing among packages. More concretely, applications are installed on a shared binary server, and then exported to all end user machines through a standard network file-sharing interface. When an application is executed, accesses to binaries, configuration files are redirected to the central binary server.

There are other frameworks which intercept system calls in windows. Whips (windows host intrusion prevention system) [9]is an open source driver which relies on the same technique (like FVM driver) of intercepting system calls by turning off write protection to System Service descriptor table(SSDT). It detects the violation of security rules by monitoring system calls made by a particular process. They enforce the rules by checking with an access control database (ACD).

Easy Hook [10] is an alternative to Microsoft Detours library. It provides an easy way to hook unmanaged code from managed environment.

Chapter 3

3 System Architecture

3.1 Overview

The aim of this project is to enable client to run applications installed on a remote server without actually installing them. The binaries (executables and dlls) are shared on the remote server. The Client opens the executable from the shared location and runs the application as if it were installed locally. Main purpose of this is to centralize patching and installing applications.

Binary Streaming leverages the Feather weight Virtual Machine architecture. The interception of system calls is done at the kernel level. This can be done by turning off the write protection to the system service descriptor table (SSDT).This is very much similar to the system call table in Linux kernel and contains function pointers to various system calls. We replace the entries to various system calls in this table.

Whenever an application starts execution, it tries to look for its operating environment various dlls it depends upon and registry keys that are created when the application is actually installed on the server. These dlls are copied and registry keys are created typically by the installer.

Turning off the write protection is done by the following code.

```
__asm {
                push eax
                mov eax, cr0
                and eax, not 10000h
                mov cr0, eax
                pop eax
        }
```

Replacing the entry to NtCreateFile in system service descriptor table.

(NtCreateFileProc)(SystemService(ZwCreateFile)) = FvmFile_NtCreateFile;

FvmFile_NtCreateFile → This is FVM's implementation of the system call NtCreateFile. *After the pointer has been replaced in the system call table, all the calls by all the processes to NtCreateFile will go through our replaced function.* But we only want our process that is running inside FVM to be redirected. All the other processes should run normally. This distinction is made by checking the pid of every process that makes the system call. If the pid is mapped to some FVM, then we redirect this system call appropriately or else we just call the original system with exactly the same arguments.

The following system calls are intercepted by the Binary streaming framework.

i) NtCreateFile
ii) NtOpenFile
iii) NtQueryAttributesFile
iv) NtQueryFullAttributesFile
v) NtDeleteFile
vi) NtOpenKey
vii) NtCreatekey

Other system calls like NtQueryInformationFile, NtSetInformationFile need not be intercepted because these functions take the open file handle as a parameter. And we are already redirecting NtCreateFile and NtOpenFile, so the file handle returned would point to the redirected file.

The code for intercepting the system calls is written at the driver level. It is compiled using the Windows driver Kit (WDK) build environment and loaded using instdrv.exe. *Appropriate build environment should be chosen depending on the operating system.*

Driver is loaded by using the following command :

instdrv.exe rnifvmdr rnifvmdr.sys
<Name of the driver>

Run the above command on the administrator command prompt. Instdrv is a win32 application which installs a device driver. We can alternatively create a service which starts whenever the system is booted up and loads the driver in the background.[8]

Applications like Microsoft Office are installed on a binary server. The WindowsVolume of the server is shared. The registry HKLM\Software hive of the server is imported to the client under the HKLM\Software\FVMAppStream hive before launching any application. This is a one time setting. A *TestKey* is created in the registry to specify the server location. (name/IPAddress).

WindowsVolume is typically the C:\ drive on Windows machines. But it need not be the case. Some machines may have windows installed on any other drive (like D:\ drive). In a dual boot scenario different versions of windows may be installed in C:\ and D:\ drives. Always the drive that contains both the windows and office installations should be shared. *It may or may not be the drive that is running the operating system that is currently booted up.*

How is an executable started in the Binary Streaming mode ?

When the user opens a shared folder on the binary server and double clicks on an executable, we have to add this process to a VM. This is done by ShellExecuteHooks.[9] We create dofshell.dll which implements the IShellexecuteHook interface which contains Execute (LPSHELLEXECUTEINFO pei) method. We can get the file path from the shell execute info pointer. Check whether the binary server name (in the *TestKey*) exists in the path or not. If it exists the executable is indeed opened from the Binary server, create the process using the API DetoursCreateProcessWithDLL() in suspended mode with C:\detoured.dll and C:\fvmdll.dll. Add the process to a VM by using an ioctl IO_CREATE_VM. The arguments to the ioctl are the serverName and the pid of the process. ServerName or IPAddress is needed as an argument because this particular process is mapped to this server and all the accesses would be redirected to this particular binary server. Once the process is added to the VM, it can be resumed using

ResumeThread (processInfo.Thread). S_OK is returned indicating that the item has been executed successfully. The shell need not process it. Shell execute hooks are registered in the following location in the registry *HKEY_LOCAL_MACHINE\ SOFTWARE \Microsoft\Windows\CurrentVersion\Explorer\ShellExecuteHooks*. This method of using ShellExecuteHooks has been deprecated in Vista and Windows 7.

With caching enabled there is a possibility of the user opening powerpoint.exe from the cache instead of the remote location. So even in this case the process is added to the VM.

In case the process is not an executable, or if it is not from the Binary Server/Cache we return S_FALSE and the shell will handle it.

When the application starts it tries to open its dependent files and registry entries. We redirect the file accesses to the Binary server and the registry accesses to the HKLM\Software\FVMAppStream hive. The only file accesses that get redirected to the client machine are the actual files created by the application i.e if the application is powerpoint, the .ppt files should be saved on the client but all the dll access should be redirected to the Binary server. This distinction is made by making the assumption that accesses to the **system folders** ("Program Files", "WINDOWS", "WINNT", "Documents and Settings\All users") should be redirected to the server and accesses to all other folders should be redirected to the client machine. Theoretically it is possible to save powerpoint or word files in system folders but this assumption is good for most of the cases.

In case of registry, all the accesses to the HKLM\Software hive are redirected to the server's hive (HKLM\Software\FVMAppStream) that has been imported to the client's registry prior to the installation. User specific settings generally go into the HKEY_CURRENT_USER hive and they are not redirected. They go to the client's registry.

3.2 Caching & Optimization

In Binary Streaming, all the dlls and configuration files are accessed over the network. This may cause huge latency in running the application every time a new dll/config file is

accessed. Server's registry hive is anyway imported before starting the application. So there won't be any additional latency involved for registry accesses.

To improve the performance of Binary Streaming, files are cached locally on the client machine. All the file accesses are first looked up in the cache, and then if it is not found they are redirected to the server.

The location of the cache is "C:\FVMAppStream". The same folder structure on the server is retained in the cache i.e C:\Program Files\Microsoft Office\Office11\Common Files\mso.dll would be cached at C:**FVMAppStream**\Program Files\Microsoft Office\Office11\Common Files\mso.dll.

This will be helpful if two different applications have a same dll name. If we have a flat cache (all dlls in one big directory) it will result in overwriting the existing dlls.

It works as follows::

i) NtOpenfile, NtCreateFile, NtQueryAttributes, NtQueryFullAttributes are redirected to the cache.

ii) When a file is accessed, it is first copied to the cache and the local copy is accessed. If the file cannot be copied for some reason, access the remote copy.

Before copying the file, the directory structure has to be created. Both the remote Path and the cache path are needed to create the directory structure. We search for the delimiter "\" and check if the directory exists or not. This is done by ZwQueryAttributes (). If the directory exists we skip and look for the next delimiter else a new directory is created by ZwCreateFile. After the directory is created the file attributes for the corresponding remote directory (on the server) are set to the local cache directory. This is done by calling ZwQueryAttributes on the remote directory and ZwSetAttributes on the local directory.

The name after the last delimiter "\" is considered as the file name. The directories are created just upto the last delimiter.

After all the directories are created, the actual file is copied. *The system calls NtQueryAttributes etc. can be called on both files and directories.* If it is a file, it has to be copied byte by byte, if it is a directory it can be created as mentioned above and the file attributes can be set. This distinction is made by specifying the FILE_NON_DIRECTORY flag in ZwOpenFile. If it returns an error, then the file is indeed a directory. If it is a normal file, we copy it by reading a buffer (2048 bytes) of data using ZwReadFile() and writing using ZwWriteFile() in the newly created file. If there is an error in opening/copying the remote file, the function returns an error code and the application accesses the remote copy directly.

iii) If the local copy already exists, we compare the change time of the local copy and the remote copy. If the change times don't match we overwrite the local copy with the remote one and access the remote copy.

The best way for checking whether a file exists or not is by using ZwQueryAttributes. But in this case we have to check for the change time if the file exists. So this check is done by using ZwOpenFile. If the file exists, the open file handle is used to get the file basic attributes by calling ZwQueryInformationFile. Failures may occur if the remote file is not accessible due to network failure. In this case we proceed with the local file without checking for the change time of the file.

If the change times of the local and remote copy don't match, delete the local copy first using ZwDeleteFile. Copy the file from the server again.

3.3 Disconnected mode

One of the purposes of caching files is to work in disconnected/ offline mode. If the server is down or network connection is not there, we must be able to run the application using the cached files.

If the network connection is lost suddenly, the application continues to work till it reaches a point where it accesses a file that does not exist in the cache. At this point the application terminates, giving an error message.

The executable is launched from the cache. The application can directly access a file from the cache. This will happen if the application uses a relative path. So it will directly try to access a dll from the cache like C:\FVMAppStream\Program files …\mso.dll. In this case we have to check if the local copy exists or not and try to open the remote copy to check for the latest version. *Even in disconnected mode we always have to query the server to check for the latest file version. The network may be up now.*

Applications read the path to dlls from the registry. The path may be something like D:\Program Files\...\mso.dll. The System Drive on the server may be "D:\". A D drive may not even exist on the local machine. So when redirecting these paths to the cache we have to rename the drive "D:" to "C:". For redirection to server we always assume that the network share name is "C". Even a D:\ drive on the server is shared as "C".

Short file names read from the registry should be expanded using the local directory structure. To expand a short file name (8.3) format to a long one, we need to open the file and use NtQueryDirectoryAttributes. In the disconnected mode, server is not available. So this expansion must be done using the directory structure of the local cache.

Working in disconnected mode is significantly faster than the connected mode Firstly because file copying is not done, the application just runs with the existing files. No version check is done for the files. Because the server is not connected we can't check for the change time of files. Each dll gets accessed multiple times. We perform a check every time the application accesses the file.

3.4 Installation

Current BS prototype works on Windows XP professional. Currently, both the server and client have the same version of windows. Applications like Office 2003 are installed on the binary server. The entire C:\ drive (system drive) is shared on the binary server. Windows XP professional supports two types of file sharing i) Simple File Sharing ii) Other mode. Simple file Sharing is disabled by default on domain joined machines. Simple File Sharing enables everyone on the network to access the files. We can get finer control by disabling simple file sharing.

The entire HKLM\Software hive of the server is exported to the HKLM\Software\ FVMAppstream. In previous versions of windows, this can be done a remotely i.e the registry of the server can be accessed from the client remotely. In windows XP this cannot be done due to security reasons. So this is done in two steps. First the HKLM\Software hive is exported to a file on the server. This file is saved in the C:\ drive. This is done as a part of the server setup. During the client setup, this file is copied to the client and imported to the HKLM\Software\FVMAppStream hive.This will actually impose a restriction that server setup has to be done before the client.

Client setup consists of the following steps

i) Check whether detours express is installed or not. This can be done in multiple ways. We are searching for some files that detours express installs during its setup in the Program files folder. This check can be done in multiple ways--looking for specific keys in the registry, searching for a GUID of some non-optional component in the msi of detours express. The msi of detours express can be opened using ORCA tool and a non-optional component can be searched for. If detours express is not installed we point to the Microsoft url for detours express.

ii) All the installation files (dlls, Client_regcopy.exe, fvmserv.exe) are copied to the C:\driver folder on the client.

iii) Takes the server location as input imports the registry as mentioned above. All the registry keys are setup as non-volatile and they survive a system reboot. It also creates a TestKey which points to the server location.

iv) It copies the actual driver file, rnifvmdr.sys to the C:\WINDOWS\System32\drivers folder. It creates a service “Binary Streaming” which starts during the boot up time and loads the driver. A VM is also created in the service by sending the ioctl IO_CREATE_VM to the driver. Any new process can just add to the VM. This will improve the startup time of the application.

v) If we uninstall the client, service is removed and the installation files get deleted.

2. Server setup consists of the following steps.

i) Export the HKLM\Software hive on the server to C:\bc.dat file.

ii) Disable simple file sharing by setting the key HKLM\system\Windows\CurrentControlSet\Lsa \ForceGuest.

iii) Share the system folder of the server using NetShareAdd API. Giving Read/write permissions to all. The share name used is always “C” irrespective of the system drive. In case there is any other folder with the same share name, an error message is displayed.

Chapter 4

4 COM Virtualization

4.1 Windows COM Architecture

Component Object model (COM) is a method for sharing binary code across different applications. COM allows reuse of objects with no knowledge of their internal implementation, as it forces component implementers to provide well-defined interfaces that are separate from the implementation. It provides a language-neutral way of implementing objects that can be used in environments different from the one in which they were created, even across machine boundaries.

To get Binary Streaming working in the windows environment, COM objects have to be handled. An application can access load a particular dll/ start an executable using the COM framework. In this section, we give a brief overview of COM.

COM programmers build their software using COM-aware components. Different component types are identified by class IDs (CLSIDs), which are Globally Unique Identifiers (GUIDs). A *GUID* (**g**lobally **u**nique **id**entifier) is a 128-bit number. GUIDs are COM's language-independent way of identifying things. Since GUIDs are unique throughout the world, name collisions are avoided (as long as you use the COM API to create them).

Each COM component exposes its functionality through one or more interfaces. The different interfaces supported by a component are distinguished from each other using interface IDs (IIDs), which are GUIDs too. To use the functionality implemented by a COM object, one needs to know the class ID and the interface ID of the particular interface that implements the required functionality.

A *coclass* (short for **c**omponent **o**bject **class**) is contained in a DLL or EXE, and contains the code behind one or more interfaces. The coclass is said to *implement* those interfaces. Each interface and coclass has a GUID.

A *COM object* is an instance of a coclass in memory. Note that a COM "class" is not the same as a C++ "class", although it is often the case that the implementation of a COM class is a C++ class.[10]

A *COM server* is a binary (DLL or EXE) that contains one or more coclasses.

We walk through an example of an AddObj class[13], which contains the implementation for a fast addition algorithm. After the implementation of the class, we describe how a client can use this COM class for performing the addition.

4.1.1 Interfaces

In C++ language, an interface is an abstract base class. It contains the prototypes of various functions. Any non-abstract class that derives from this base class, has to provide the implementations for all the functions in the base class.

AddObj class contains the implementation for a fast addition algorithm. It implements the IAddObj interface which has three methods.

```
class IAddObj {
public:
 virtual ~ AddObj();
 virtual void SetFirstNumber(long x) = 0;
virtual void SetSecondNumber(long y) = 0;
virtual void  AddResult(char *buff) = 0;
};
Class AddObj : public IAddObj {
  // Actual implementations of the above 3 functions …
}
```

Every COM class must implement the IUnknown interface. It contains three methods.

i) AddRef – for incrementing the reference count.

ii) Release – for decrementing the reference count.

iii) QueryInterface – Takes the interface id as an argument and returns a reference to the appropriate interface that is requested by the caller. A coclass can implement many different interfaces.

4.1.2 Class Factory

As per COM guide lines, every COM object must have a separate implementation of the interface IClassFactory. Clients will use the factory class to get a reference to an instance of our implementation.

Using a factory class one can get finer control over the creation of the objects. We can control various aspects like the number of number of instances of the object etc.

IClassfactory inherits from IUnknown and contains the CreateInstance method.

CAddFactory is the factory class for our AddObj component.

```
CAddFactory:: CreateInstance(IUnknown* pUnknownOuter, const IID& iid, void** ppv)
{
/* ….. */
// Create an instance of the component.
  CAddObj* pObject = new CAddObj ;
//  Get the requested interface using QueryInterface
    return pObject->QueryInterface(iid, ppv) ;
/*… */
}
```

4.1.3 Interface Definition Language

The Microsoft Interface Definition Language (MIDL) defines interfaces between client and server programs. Interfaces have to be defined in IDL and they have to be compiled

using MIDL which will generate a header file corresponding to the required language and a type library.(TLB). The binary metadata contained within the type library is meant to be processed by language compilers and runtime environments (e.g. VB, Delphi, the .NET CLR etc.). The end result of such TLB processing is that language-specific constructs are produced that represent the COM class defined in the .TLB (and ultimately that which was defined in the originating IDL file).

The following user level APIs are intercepted using the Detours library.

i) CoGetClassObject

ii) CoRegisterClassObject

iii) CoCreateInstance

iv) CoGetInstanceFromFile

A new process is created using the detours API **DetoursCreateProcessWithDll** (..”fvmdll.dll”). fvmdll.dll contains hooks for intercepting the user level APIs.

4.2 Independent Process COM

An Independent process or out of process COM server (EXE) is launched as an independent application. CoCreateInstance is called with a CLSID X. RPCSS looks up in the registry for this CLSID under the HKLM\Software\Classes\CLSID\X (or HKCU). If there is a subkey LocalServer32 then it is an Out of Process COM server. The process can be launched on a local machine or a remote machine. We are dealing with the case when the COM server is launched locally. This subkey contains the full path to the EXE. RPCSS launches this executable as a separate process. The COM server (EXE) registers itself with OLE using CoRegisterClassObject so that other applications can connect to it.

The HKLM software hive of the BS server is imported to the HKLM\ Software\ FVMAppStream\Software hive of the BS client. We call this imported registry on the client as the **server’s registry** hive. An application running inside FVM on the client calls CoCreateInstance with a CLSID X.

Independent Process COM Virtualization works as follows

Lookup for the CLSID in the server's registry hive. (HKLM\ Software\ FVMAppStream\ Software\Classes\CLSID\X). If it has a subkey LocalServer32, then it is an Independent Process COM server. The subkey contains full path to the executable.Generate a new CLSID Y and create a new key in the host's registry. (CoCreateGuid) (HKLM\ Software\Classes\CLSID\Y). Clone the branch X to Y.

LocalServer32 contains the full path to the executable that will be launched as an independent process. We want this process to run inside FVM. If RPCSS launches this process we cannot add it to FVM. So the LocalServer32 key is modified to point to a wrapper executable for which the command line argument is the actual COM server. RPCSS will launch this wrapper executable with the command line argument as the COM server. Our wrapper executable will use CreateProcess to launch the COM server and add its pid to the FVM driver using DeviceIoControl (ioctl).

CoCreateInstance is called with the new CLSID Y that is generated above. The registry keys that are generated above are created in VOLATILE mode. They are created on demand and get erased once the system reboots. The mapping between X→Y is stored in the registry under HKLM\ SOFTWARE\ rnifvm\ COMV\ FVM<*ID*>*{X}*

This typically looks like …FVM0{8BC3F05E-D86B-11D0-A075-00C04FB68820}

FVM0 has a mapping for CLSID {8BC3F05E-D86B-11D0-A075-00C04FB68820}.

Generation of a new CLSID will take care of possible CLSID clashes. When the newly launched COM server registers itself with its original CLSID using CoRegisterClassObject, that call is intercepted and we register it using the new CLSID Y that is generated. The new CLSID is read from the registry entry that contains a mapping between X & Y.

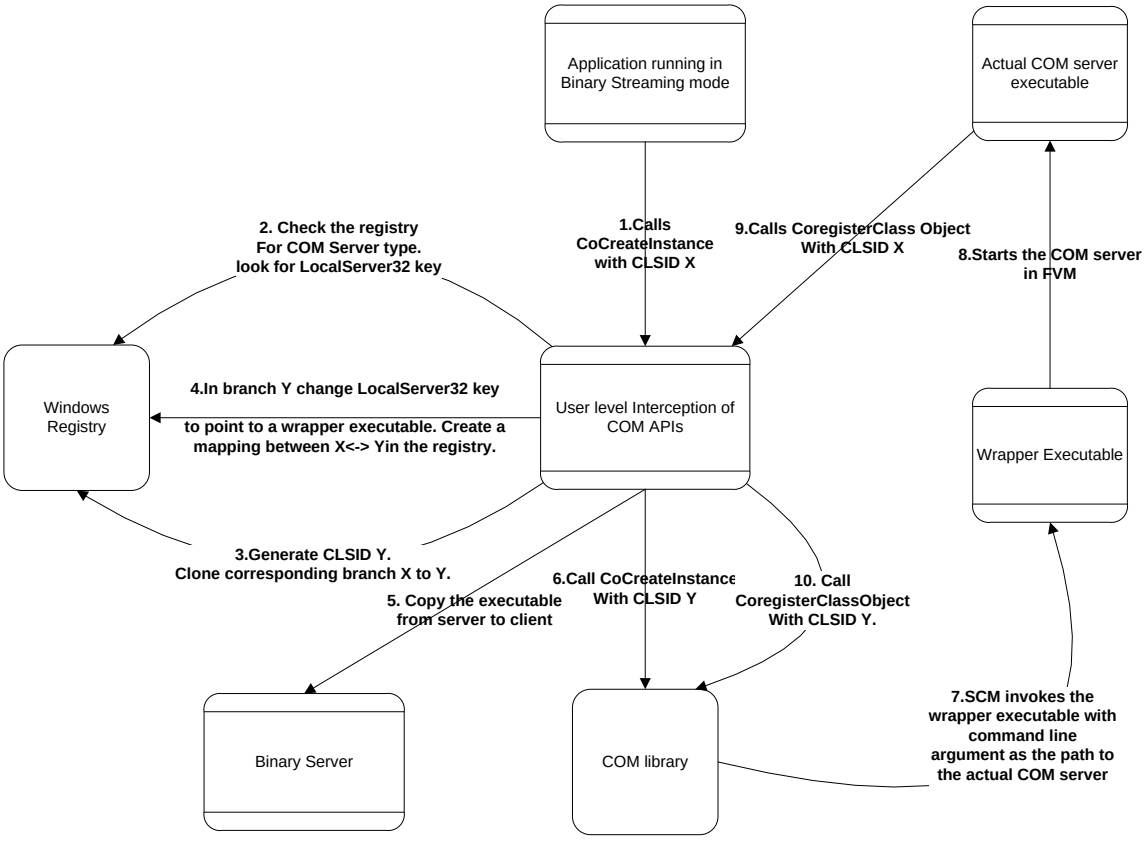

Figure 2: Independent Process COM Virtualization

4.3 Inprocess COM

An Inprocess COM server is a dll that obeys a certain protocol. It has to export four methods DllGetclassObject, DllCanUnloadNow, DllRegisterServer, DllUnregisterServer methods. Clients will use DllGetClassObject method to get an instance of the Classfactory through which they get an instance of the actual coclass.

To load a dll using the Inprocess COM model, a process calls CoCreateInstance with a CLSID id X. The COM library looks up in the registry (in both HKLM and HKCU hives with preference to HKCU) for the key HKLM\Software\Classes\CLSID\{X}. For InProcess COM dlls, typically there will be two subkeys (InprocServer32 and ProgId). The InprocServer32 key contains the **full path to the dll** and the **Threading model** (Apartment or Single Threaded). COM library loads the dll into the address space of the

process, and returns a pointer to the interface requested by the process. The process can access all the methods of the interface implemented by the dll using this pointer.

Before loaded getting loaded by a COM client, the dll has to be **registered**. *Registration* is the process of creating registry entries that tell Windows where a COM server is located. *Unregistration* is the opposite - removing those registry entries.

DllRegisterServer … This function adds registry keys mentioned above (HKLM\ Classes\ CLSID\{X}) which can be looked up by the COM library for loading the dll.

DllUnregisterServer ..This function removes the keys from the registry. This is typically done when the application is uninstalled.

DllGetClassObject.. Clients call this method to get an instance of the factory class.

```
DllGetClassObject (const CLSID& clsid, const IID& iid, void** ppv) {
 //Check if the requested COM object is implemented in this DLL
//There can be more than 1 COM object implemented in a DLL
 if (clsid == CLSID_AddObject) {

   //iid specifies the requested interface for the factory object
    //The client can request for IUnknown, IClassFactory

    CAddFactory *pAddFact = new CAddFactory;
      return pAddFact->QueryInterface(iid , ppv);
  }  else
       return CLASS_E_CLASSNOTAVAILABLE;
}
```

There is a command line tool **regsvr32** which registers/deregisters Inprocess COM dlls.

Let us consider the example of AddObj.dll . It registers itself with the CLSID {92E7A9C2-F4CB-11D4-825D-00104B3646C0} (call this X) which is a 128 bit globally unique identifier (GUID). This value is hardcoded inside the dll.

The following keys are added to the registry by DllRegisterServer method.

"HKCU\CLSID\X\InprocServer32" → C:\AddObj\AddObj.dll (This key contains the full path to the dll).

"HKCU\CLSID\X\ProgId" → FastAddition

HKCU\FastAddition \CLSID →X

A Client executable wants to utilize the addition functionality implemented in Addobj.dll. It has to call a method that is implemented by the IAdd interface of AddObj .dll. Now it calls CoCreateInstance with the following arguments.

```
CoCreateInstance(                                    CLSID_AddObj,
                  NULL,
                  CLSCTX_ALL,
                  IID_IAdd,
                 (void**) &pFastAddAlgorithm )
```

CLSID_AddObj→ X, the same GUID with which AddObj registered itself previously.

IID_IAdd – Interface ID of AddObject.

If CoCreateInstance call succeeds we call the methods in the interface. SetFirstNumber is one of the methods implemented by this interface.

```
((IAdd *) pFastAddAlgorithm)->SetFirstNumber(n1);
```

pFastAddAlgorithm will be NULL in case that particular interface is not implemented by the COM server.

How does Inprocess COM virtualization work in Binary Streaming?

CoCreateInstance call is intercepted using the Detours framework. Look up for the CLSID in the server's registry hive (HKLM\ Software\ FVMAppStream\ Software\ Classes\CLSID). Determine the type of COM server. If it is Inprocess COM server (contains the InprocServer32 key) then proceed further.

Using the CLSID we can get the path to the dll from the registry. Translate the path to point to the location of the dll on the server. (C:\Program Files\Office\A.dll → \\binserverXp\c\ Program Files\Office\A.dll).

Now the dll has to be copied to the corresponding location in the BS cache. This can be done either from the user level or at the driver level. Since the code for caching has been implemented completely at the driver level, this was file copy was done by issuing a NtqueryAttributes call. The driver intercepts this syscall, looks up for the file in BS cache. If it is not present it copies the file to the cache and in the process creating the required directory hierarchy as well.

The dll has to be registered in the **client's registry.** This is generally done by the installer when the application is installed using tools like **regsvr32**. We use a generic program (registerdll) which does the registration. It takes the path to the dll as a command line argument and does the following:

i) LoadLibrary(<*Full Path to the dll*>)

ii) GetProcAddress("DllRegisterServer")

iii) Call DllRegisterServer using the proc address.

This registerdll process is created by using the **CreateProcess** call. When we create a new process from inside a process running in FVM, even that process gets added to FVM. But registerdll process should not be intercepted. Otherwise the registry keys get created in the server's registry hive under the FVMAppStream key. **COM library loads the dll only if it reads the key from the client's registry hive.**

There is a call back function at the driver level, which notifies it if a new process is created (PsCreateProcessNotify). All the pids of the processes running inside FVM are stored in a global data structure (array). To create an exception for this registerdll process, we use an ioctl that will add the pid of the process to an exception array. This has to be done before the process starts execution. So the process has to be first created in **suspended mode** and the execution has to be resumed after the ioctl call is made to the driver.

The main thread has to wait for the registerdll process to complete and execution should be resumed only after that. Otherwise, the main thread will go ahead and perform the CoCreateInstance call without the dll getting registered. This may lead to a **race condition.** So we use WaitforSingleObject function in the main thread to allow for the registration to complete.

```
WaitForSingleObject (piProcessInfo.hProcess, INFINITE);      //wait indefinitely
```

After registration, CLSID X is present in both **the server's and the client's registry hive**. Now if we call CoCreateInstance with CLSID X, COM library looks up for the CLSID in the client's hive. But due to registry redirection logic in the driver, it gets redirected to the server's hive where it finds X (HKLM\Software\Classes\CLSID\X → HKLM\ Software\ FVMAppStream\Software\Classes\CLSID\X).This redirection will defeat the entire purpose of registering the dll.(COM library can load the dll only if it reads from the client's registry). So we have to make sure that COM library cannot find X in the server's hive (so it will be redirected back to the client's hive at the driver level).

There are multiple ways to achieve this. We can add a rule at the driver level … all access to HKLM\..\FVMAppStream\Software\Classes\CLSID should be redirected back to the client. A better way is to rename the key in the registry (change X→1X). Now COM library won't be able to find X in the server's registry and gets redirected back to the client's registry where it will find X. This means we have to look for X as well as 1X in step 1 where we determine the type of COM server. When we access the registry hive to determine the type of COM server, we introduce a special marker in the path (?FB?) to tell the driver that the registry access should not be redirected. When COM library looks up in the registry it always redirected because there is no marker in the path.

After all the above steps have been performed, call CoCreateInstance with the original CLSID. The CLSID of a dll is hardcoded in the dll itself. It registers itself with this CLSID.

The above design works if the same CLSID is not present (previously registered by some other application) on the client. A CLSID is a GUID which is a 128 bit number (expected

to be unique). *But if an older version of the application is already installed on the BS client, there may be a clash of CLSIDs.*

Let us consider the following scenarios where the CLSIDs clash.

Both the application that is running in FVM as well as the original application installed on the client is supposed to run. If the CLSIDs are of different type, we cannot make this happen. For eg. If the CLSID installed on the client corresponds to an Independent Process COM server and the one on the server is an Inprocess COM dll then there is no way the above design will work because the registration process will overwrite the existing key. Even if both CLSIDs correspond to InProcess COM servers, the threading model has to be the same.

We make the following changes to the design for taking care of CLSID clashes.

Before registering the dll, look up in the registry (both HKLM & HKCU hives) for CLSID X. If it is already present, **do not register the dll.** This will ensure that the original application that registered this dll will work correctly.

If CLSID X corresponds to an Inprocess COM server with the same Threading Model, then we have to somehow tweak the location of the dll in the **InprocServer32** key, so that COM library will load the dll corresponding to the original application when it is not running inside FVM and the server's dll when it is running inside FVM.

HKLM\Software\Classes\CLSID\X\InProcServer32 → path A (path to the dll on the client)

HKLM\Software\FVMAppStream\Software\Classes\CLSID\X\InProcServer32 → path B (path to the dll on the server)

Path C → Corresponding BSCache path to A. (C:\FVMAppStream\<*path A*>\)

Path D → Corresponding server path to B. (\\binserverxp\c\<*path B*>\)

The dll has to be copied from location D to C. CopyFile() API is used for this purpose.

Now when the COM library loads a dll from location A, if the process is running inside a FVM it gets redirected to location C from where it loads the server's dll. If the process is not running inside FVM, COM library normally loads the client's dll from location A.

The names of the dlls at paths A & B may be different. So when we copy the server's dll, it has to be renamed to the client's dll. This is because the InprocServer32 registry key points to the client's dll.

4.3.1 Problems encountered in the design

In the initial design of Inprocess COM virtualization, we decided to generate a new CLSID Y and copy the contents of the server's registry key (HKLM\ Software\ Classes\ CLSID\X) to the client (HKLM\Software\Classes\CLSID\Y). Change the path in InprocServer32 key to point to the server's dll in BSCache. Now call CoCreateInstance with CLSID Y. The intent was to avoid CLSID clash. But when we call CoCreateInstance with this CLSID Y, it is returning a non-standard error code. The DllGetClassObject method of the Inprocess dll matches its own CLSID with CLSID requested by the client. It will proceed further if and only if the CLSIDs match. SysInternals tools like Listdlls will show that the dll is actually loaded by the process but CoCreateInstance never returns a successful error code.

A confusion initially was whether the dll is loaded by COM library or RPCSS. In the former case the lookup for the CLSID takes place in the process context, so lookup in the client's registry hive is redirected to the server's hive. We tried to modify the InprocServer32 key in the server's registry hive to point to the dll on the server (\\binserverxp\C\Program Files...\A.dll). So COM library should be able to locate the dll directly and load it. Even this is not working correctly and is giving a non-standard error code.

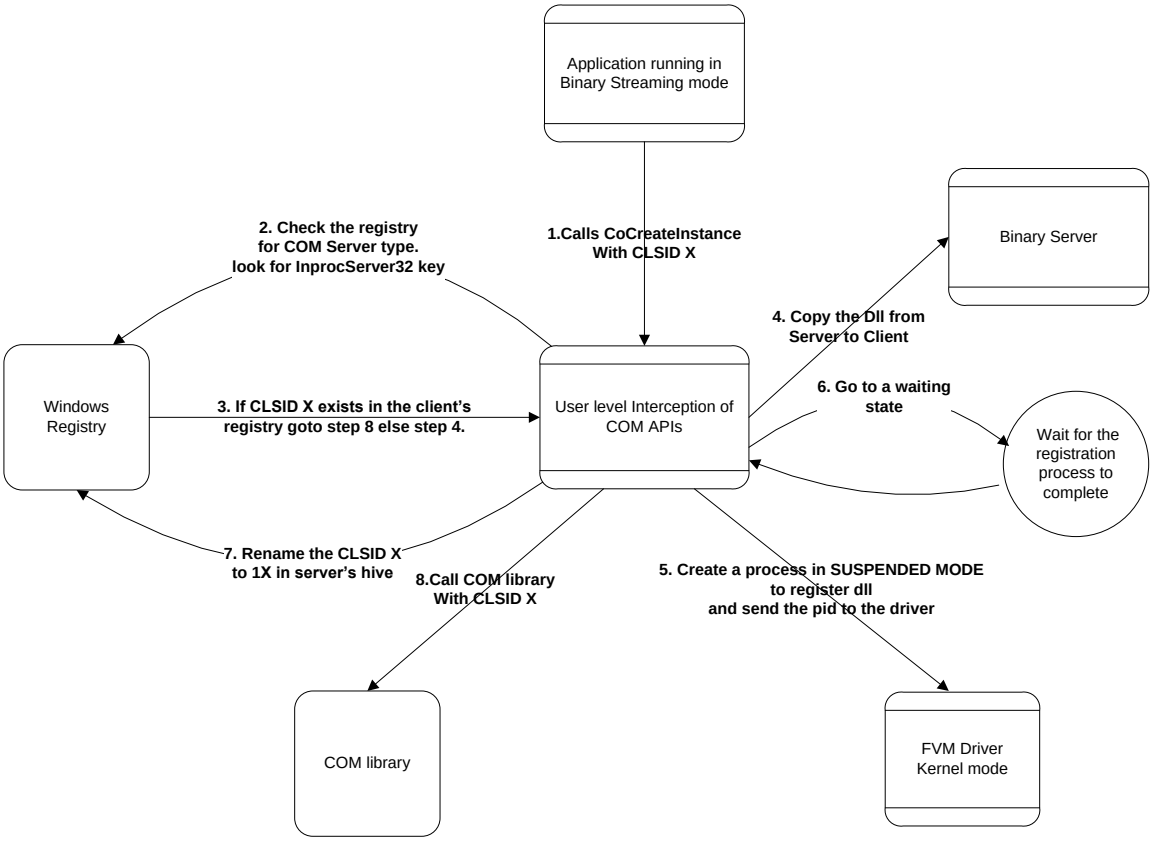

Figure 3: Inprocess COM Virtualization

.

Chapter 5

5 Porting FVM to Vista & Windows 7

In the previous versions of FVM, system calls were intercepted by turning off write protection for SSDT (System service Descriptor table). This is accomplished by switching a bit in CR0 (Control register 0) and resetting the bit after changing the pointers in SSDT.

The same technique works for 32 bit vista SP1, SP2 and windows 7. For 64 bit versions Microsoft has used a PatchGuard to prevent unauthorized patching of SSDT. **The above technique will not work for 64 bit versions.**

The system call ordinals are different for various versions of windows. We have disassembled ntdll of vista & win 7 and analyzed it in IDE pro. We can get the ordinals of the system calls directly after disassembly.

For Eg:- For NtCreateEventPair the disassembly will look like ..

```
mov eax, 0x3b
```

The system call ordinals for vista and win 7 are listed below.

System Call	**Vista**	**Windows 7**
ZWRESUMETHREAD	0x130	0x130
NTCREATETHREAD	0x4e	0x57
NTFSCONTROLFILEPROC	0x96	0x86
NTQUERYDIRECTORYOBJECT	0xdb;	0xe0
NTSETVALUEKEY	0x0144	0x166

NTSECURECONNECTPORT	0x011e	0x138
NTCREATEWAITABLEPORT	0x0073	0x5e
NTCREATEPORT	0x0047	0x4d
NTOPENEVENTPAIR	0x00b9	0xb2
NTCREATEEVENTPAIR	0x3b	0x41
NTOPENIOCOMPLETION	0x00bb	0xb4
NTCREATEIOCOMPLETION	0x003d	0x43
NTOPENTIMER	0x00cc	0xc9
NTCREATETIMER	0x4f	0x59
NTOPENSEMAPHORE	0xc6	0xc3
NTCREATESEMAPHORE	0x4c	0x55
NTOPENMUTANT	0xbf	0x4a
NTOPENPROCESS	0xc2	0xbe
NTCREATEMUTANT	0x43	0x4a
NTCREATEMAILSLOTFILE	0x42	0x49
NTCREATENAMEDPIPEFILE	0x44	0x4b
NTQUERYFULLATTRIBUTESFILE	0xdf	0xe4
NTQUERYATTRIBUTESFILE	0xd4	0xd9

Table 1: System call table

The ordinals for common system calls like ZwCreateFile, ZwOpenFile, ZwQueryInformationFile, ZwSetInformationFile, ZwQueryVolumeInformationFile, ZwDeleteFile, ZwCreateEvent, ZwOpenEvent, ZwCreateSection, ZwOpenSection,

ZwClose, ZwOpenKey, ZwCreateKey, ZwqueryKey, ZwDeleteKey, ZwEnumerateKey etc are extracted from function pointers. There is a check in the DriverEntry function which checks for the windows version and assigns the appropriate system call ordinals depending on the version.

if ((**IoIsWdmVersionAvailable**(6, 0x00)) // Major version 6 Minor version 0 ..
{

// assign ordinals for vista.

}

The system calls ordinals for Vista SP1 and SP2 are the same.

Most of the common system calls involving file and registry operations are invoked by explorer and they can be easily traced in DbgView. The uncommon ones like NtCreateEventPair, NtConnectPort are tested by using a user level program which explicitly calls these functions. I have declared the required data structures like OBJECT_ATTRIBUTES, UNICODE_STRING etc. and I have written a program which calls all the system calls and tests whether they are working or not. For some system calls where I know the corresponding user level API, I have called the user level API. For others I did a LoadLibary on ntdll and used GetProcAddress on the system call.

The application name is accessed through the EPROCESS structure in the driver. The offset to the process Environment block in the EPROCESS structure is hardcoded. It is different for various operating systems like Vista & Windows 7. Giving a wrong offset results in a blue screen.

EPROCESS --> PEB (Process Environment Block) -->ProcessParameters--> Application name

struct EPROCESS has a member PEB which is accessed using its offset in the structure. This offset is different in Vista (0x188 instead of 0x1b0 in XP). Got the correct value using Windbg which dumps the entire structure with offsets (dt EPROCESS).

5.1 Known Issues in testing Windows 7

1. Dump files are not getting generated when windows 7 RC is run on VMWare workstation 6.5.3 version. We can enable kernel memory dump by going to Advanced system settings → Startup & Recovery → System Failure tab. Even after doing these steps no kernel dump is generated. The workaround here is to install Windows 7 on Sun's Virtual Box.

2. Dump files automatically get deleted when no error report is sent. *Before cancelling the error report, copy the dump file to another location for further analysis using windbg.*

Chapter 6

6 Evaluation

6.1 Correctness testing

The Binary Streaming prototype has been tested on Windows XP professional and Microsoft Office 2003. Most of the testing is done on Word, Excel and Powerpoint applications.

Some features like office assistant are not working in Binary Streaming mode. We can create ppt, word and excel files and save them on the BS client locally without any problems. Double clicking on a ppt file on the client won't work because .ppt extension is not associated with any application on the client. Powerpoint has to be started first using BS, and the required file has to be opened from the menu, File→ Open.

Word and Excel give error messages when they are started. But pressing OK button will remove the dialog and word/excel opens normally.

Copying an excel sheet to MS word is not giving an option to embed the graph as an excel object. It is just getting embedded as an image.

6.2 Performance Evaluation

The startup times of applications running in Binary Streaming is compared to their startup times on a client where the application is actually installed. We compare the startup times of Microsoft Word, Powerpoint and Excel applications.

The startup time is measured as the amount of time elapsed between starting the executable and the time when the window shows up. This is a reasonable metric to measure the latency involved in Binary Streaming. The actual startup time is measured by PassMark timer utility.[15]

For Binary streaming, this is the latency for the very first execution of the application when there are no files in the cache. Note that execution times vary depending on

machine to machine. The times are measured on machines with similar configurations and the startup time is averaged over a number of iterations.

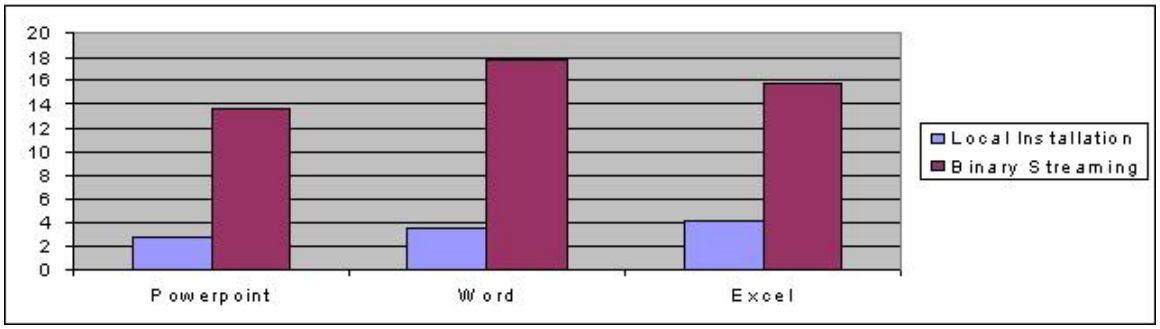

Figure 4: Performance Analysis

Bibliography

[1] Yang Yu, Hariharan Kolam Govindarajan, Lap-Chung Lam and Tzi-cker Chiueh, "Applications of a Feather -weight Virtual Machine", to appear in Proceedings of the 2008 International Conference on Virtual Execution Environments (VEE 08), March 2008

[2] Yang Yu, Fanglu Guo, Susanta Nanda, Lap chung Lam, and Tzi cker Chiueh. A Feather-weight virtual machine for windows applications. In Proceedings of the 2nd International Conference on Virtual Execution Environments, June 2006.

[3] Microsoft Corporation. Technical overview of windows server 2003 terminal services. http://download.microsoft.com/download/2/8/1/281f4d94-ee89-4b21-9f99accef44a743/TerminalServerOverview.doc, January 2005.

[4] Microsoft Detours library http://research.microsoft.com/en-us/projects/detours/

[5] Linux V-Server, http://en.wikipedia.org/wiki/Linux-VServer

[6] VMWare ThinApp for Application Virtualization, http://www.vmware.com/products/thinapp/

[7] X Windowing system -- http://en.wikipedia.org/wiki/X_Window_System

[8] Software as a Service, SaaS http://en.wikipedia.org/wiki/Software_as_a_service

[9] WHIPS - Windows Host Intrusion Prevention System http://whips.sourceforge.net/

[10] EasyHook - The reinvention of Windows API hooking http://easyhook.codeplex.com/,

[11] Instdrv – A win32 app that installs a driver. http://www-user.tu-chemnitz.de/~heha/viewzip.cgi/hs_freeware/gerald.zip/INSTDRV.C?auto=C

[12] Shell Execute Hooks -- What were ShellExecute hooks designed for? http://blogs.msdn.com/oldnewthing/archive/2008/09/10/8938051.aspx

[13] Step by Step COM tutorial – codeguru http://www.codeguru.com/cpp/com-tech/activex/tutorials/article.php/c5567

[14] CodeProject, COM tutorial http://www.codeproject.com/KB/COM/macrotopomain.aspx

[15] Passmark Timer utility … http://www.passmark.com/products/apptimer.htm

[16] Application Streaming … http://en.wikipedia.org/wiki/Application_streaming