

# Dynamic Malware Analysis on Feather weight Virtual Machine

Sushma Uppala, Yamini Pradeepthi Allu  
Experimental Computing Systems Laboratory  
Stony Brook University Computer Science Department  
Stony Brook, NY 11794-4400  
{suppala, yaminia}@cs.stonybrook.edu

---

## Abstract

The aim of the project is to implement System call and API logging and a Log Analyzer to detect malicious programs on a Feather Weight Virtual Machine. The system call and API logging is implemented by intercepting NT System Calls and Windows API. These logs are further analyzed by the FVM Log Analyzer which compares the logged behavior to existing malware behavior hence detecting any malicious activity in the Virtual Machine.

Keywords. Virtual Machines, System Call Interception, Malware analysis, Dynamic Malware Analysis.

---

## 1. INTRODUCTION

One of the major benefits of virtual machines is that they provide isolation between different instances of the virtual machine and between a virtual machine and the host environment. Isolation is the reason why security of applications running in a virtual environment is superior to that of those that run on a native system. Recently virtual machines or emulators are being used as platforms for malware analysis given the security in VMs. The security of the system becomes more important if the analysis of malware is done dynamically, i.e. the malicious program is actually run on a system to determine and analyze its behavior since it can lead to many potential problems. The program by its nature could destroy or corrupt the host systems data and state if the isolation provided by the VM is weak. Hence Virtual machine implementations that provide strict isolation are a cheap and effective solution to analyze malware behavior.

The Feather-weight Virtual Machine [1] or the FVM is one such Virtual Machine implementation which provides strong isolation and program confinement. The FVM provides both isolation with respect to processes running in different virtual machines and also isolation of a VM with respect to the host environment. Different processes on a system communicate with each other using Inter Process Communication mechanisms. FVM confines any communication between processes that belong to different VMs and processes that belong to the host environment. The various IPC Mechanisms provided by windows include Clipboard, COM, Data Copy, DDE, File mapping, Mail slots, Pipes, RPC and Windows Sockets. The FVM implements virtualization of Files, Objects and Network interfaces which confines communication through most of the above specified IPC Mechanisms. Communication messages through window messages are further confined by

## Dynamic Malware Analysis on Feather Weight Virtual Machine

intercepting system calls related to message passing and by blocking any messages that cross the VM. Communication through COM or through clipboard can be confined by virtualizing them for each VM.

Apart from confining communication between different processes, FVM also prevents data corruption and denial of service to processes in the VM by confining their write access. Data theft is also prevented by allowing only a subset of the host environment visible to the processes in the VM. Hence the FVM, which provides strict isolation, can safely be applied to isolate the execution of malicious or untrusted programs from the host environment.

Given a secure and isolated environment to run malicious programs, the dynamic analysis of malware requires logging of the actual behavior of the program when it is run on the system. This project aims at adding functionality in the existing FVM to log system calls and implement an analyzer that analyzes the logs to recognize any patterns of malicious behavior and hence preventing users from committing corrupt state of the VM to the host machine. In the following section we provide a background on the FVM and basics of Dynamic Malware Analysis. Section 3 provides the overall design of the system. Section 4 presents the limitations of current implementation and future work.

## 2. BACKGROUND

### 2.1 Feather Weight Virtual Machine

The Feather weight Virtual Machine (FVM) is a Windows based OS-level virtualization architecture [2] designed to reduce invocation latency for a VM and to scale to a large number of VMs. The FVM implements its Virtualization layer called FVM layer at the OS System call interface, which achieves virtualization through renaming of resources. FVM allows sharing of resources and adopts a Copy on Write policy, thus reducing the amount of resources that need to be allocated for a new VM. FVM achieves virtualization of Files, registries, objects and network by intercepting system calls at different level. FVM also provides strict communication confinement hence provides isolation of the VM from its host environment and from other VMs. In addition to providing flexibility, a goal of FVM is to provide fault tolerance. Hence system call logging and analysis will allow FVM to monitor malicious activity in the VM and to finally selectively commit only legitimate state changes.

## 2.2 Malware Analysis

Malware analysis is a term used for analyzing the behavior of a malicious program. The analysis could be done in many ways. The most common and established way of analyzing and detecting malware had been static malware analysis. In static malware analysis the behavior of the program is analyzed by studying the code without actually executing it. This method is useful to analyze all the paths of the code that execute very rarely. But the major disadvantage of this method is that it is very difficult to analyze behavior of large programs.

Another method of analyzing malware behavior called Dynamic Malware Analysis [3] [4] is by studying the program as it executes. Given this approach it is difficult to analyze all the paths of large or complex program. However, it is a fast and accurate effective way to analyze malware behavior. Dynamic Malware Analysis can be done in two different ways. One way is to take a snapshot of the system state (files, registries objects etc) before and after running the malware, and compare them. This approach could fail when the underlying malware behavior does not involve making any permanent state changes to the system, like creating a file and then deleting it later, or collecting email addresses from files on the system and sending emails. Hence it is effective to monitor the malware's behavior while it is executing, for example monitoring the program behavior by using tools like debuggers. In this project, we achieve monitoring of malware behavior through logging sequence of calls made by malware and later analyzing it.

## 3. DESIGN AND IMPLEMENTATION

The implementation of the system to perform malware behavior logging and analyzing can be explained as three main components. The System Call logging module in the kernel intercepts and logs all system calls related to Files, Registries and Objects. The network related API calls are intercepted and logged in a user level Layered Service Provider module. The logs generated by these modules are taken and analyzed by a user level program FVM Log Analyzer. Figure 3.1 shows the architecture of the system. The following sub sections discuss the design and implementation details for each of these modules.

## Dynamic Malware Analysis on Feather Weight Virtual Machine

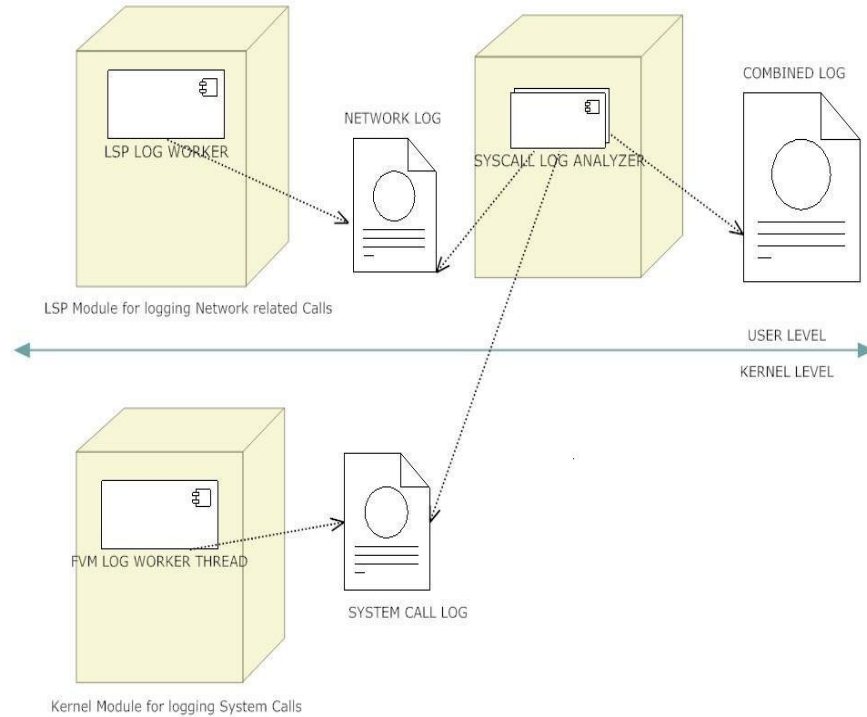


Figure 3.1. Architecture Diagram showing the System call Logging and Log Analyzer Modules

### 3.1 System Call and API Logging

Malware typically exhibit behavior involving operations related to registries, files, processes and networking on a system. On a windows machine, to perform any of the above activities, one of the related exported API or system call interface is to be used. Hence intercepting these calls and logging the arguments and return values would help in analyzing the malware behavior correctly. Windows provides API abstraction at different levels. The WIN32 API and NT Native API, each of which provides abstraction to the Windows System Calls. It can be seen that many of the APIs resolve eventually to the same system call. Hence more APIs need to be intercepted when compared to intercepting system calls to capture the same behavior. Apart from this, it is more reliable to intercept system calls, since a program at user level can bypass the WIN32 API and call the system calls directly, in which case some of the behavior is lost. However, it is easier to analyze API logs, since a high level operation may resolve into a sequence of system calls, and analyzing system call logs to arrive at the high level behavior is more complex.

## Dynamic Malware Analysis on Feather Weight Virtual Machine

The system call logging is done in two separate modules, one in the kernel level and the other in the user level. The kernel level module intercepts calls related to processes, registries and files. The user level module intercepts and logs API calls related to the network. The system calls intercepted cover most of the common malware behavior.

### 3.1.1 Kernel module for logging system calls

The kernel module for logging system calls intercepts the following system calls and logs their respective arguments.

Process related System Calls	NtCreateSection NtMapViewOfSection NtWriteVirtualMemory
Registry related System Calls	NTCreateKey NTEnumerateKey NTSetValueKey NTQueryValueKey NTDeleteValueKey
File related System Calls	NtCreateFile NtReadFile NtWriteFile NtSetInformationFile
Driver related System Calls	NtLoadDriver

Table 3.1 List of System Calls intercepted in the kernel module

The NT System Services are hooked by locating the System Service Dispatch Table (SSDT) and changing the function pointers to point to FVM Hook Services. The KeServiceDescriptorTable entry in the NTOSKRNL contains the base for the System Service Table. The function pointer to the system call is obtained by indexing the System Service Table with the Service Number for the system call. This reference is stored and the system call is called once the required action is performed in the Hook function.

## Dynamic Malware Analysis on Feather Weight Virtual Machine

In each system call that is intercepted, a log record is created with the input arguments with a time stamp attached. The performance counter is queried by a call to KeQueryPerformanceCounter to obtain the time stamp. This is a highest resolution timer on the system. This log record is added to the log queue. A worker thread (FVM\_Log\_Worker) which is a kernel thread waits on this queue and processes each log record and logs the information in a file. Hence every call to any of the above listed system calls results in a record getting added to the log file. Figure 3.2 shows the structure of a log record for a system call. Each argument of the system call is separated by a ' '.

Time Stamp	PID	TID	CALL ID	Return Value	Arguments
020d160e	200	1156	2B	0	C:\fvmBin\fvmdll.dll?96

Figure 3.2 Structure of a log record per System Call

Since the interception and logging adds additional overhead to actual processing time of the FVM, the arguments are logged into the system call log file in raw format and much of the processing is left for the analyzer which will be a stand-alone application.

### 3.1.2 User module for logging APIs

The user module is implemented as a Layered Service Provider and when loaded, intercepts the calls to network related API and logs the corresponding arguments.

The following APIs are intercepted in the LSP module.

Network Related API	WSPConnect WSPRecv WSPSend
---------------------	----------------------------------

Table 3.2 List of Network Related API that are intercepted in the user level module.

Similar to the module in kernel level, the arguments of an API are not written to the log file directly. Instead a log record is created and appended to a log queue. A user thread waits on the queue and writes these records to a network log file with a time stamp. The time stamp is obtained by obtaining the number of ticks through a call to QueryPerformanceCounter. This returns the elapsed ticks on the system. Since this is the most precise timer value on the system, both the system and network logs can be combined based on the time stamp to arrive at a per process log with a

## Dynamic Malware Analysis on Feather Weight Virtual Machine

record per system call/API arranged in sequence based on the time stamp. This merging of logs is implemented in the Log Analyzer.

In order to improve the performance of logging the various arguments are directly logged into the file, and further processing of the arguments is left to the Analyzer. For example, the host name, IP address and port number of the target is obtained from the network address that is logged when a WSPConnect API is called to connect to a port.

### 3.2 FVM Analyzer

The FVM Log Analyzer uses the logs generated by the kernel and user level modules and combines these logs to produce a per process log. Each per process log is then analyzed to determine the high level behavior of the process.

This processing is done in four phases

Preprocessor, Dependency Analyzer, Function Normalizer, Behaviour Analyzer.

**Preprocessor:** The preprocessor processes the system and network logs combines them based on the time stamp and generates a per process log.

**Dependency Analyzer:** The Dependency Analyzer processes the per process log sequentially to determine any dependencies between consecutive function calls and generates sets of function call sequences. Dependencies between function calls can be determined by looking at the function call arguments.

**Function Normalizer:** The function normalizes similar function calls to a normalized function call and also removes function calls that are not important. This simplifies the task of behavior analyzer

**Behavior Analyzer:** The behavior analyzer parses the output generated by function normalizer per each process and matches the sequence with stored templates to output behavior. The configuration for the templates can be created by looking at various common malware behavior. Figure 3.3 shows the typical final behavior log for a process.

## Dynamic Malware Analysis on Feather Weight Virtual Machine

```
*****
Process ID: 776, parent process ID: 200
Process image name: C:\createprocess.exe
*****

-- Created b.txt
  in System32 Folder C:\WINNT\system32\drivers\etc\hosts\
-- Created process Image name: C:\copy.exe
```

Figure 3.3 Example content of behavior log for a process.

The following analysis is done in the FVM Analyzer to identify malicious behavior.

### 1. Creating or ending a process:

Typical malware behavior includes starting new processes, starting/ending certain system processes and services. Such behavior is detected and logged by the FVM Analyzer as follows.

**Create Process:** When an existing program creates a process, the NtCreateSection system call is invoked. Since the System call logger intercepts and logs a record for this system call, the system call log will have an entry representing this. The behavior analyzer checks for this behavior by identifying the log entries for the System call ID of NtCreateSection. If an entry for NtCreateSection system call is found with the argument representing an executable file a corresponding entry representing this behavior is added to the behavioral log for that process.

**Terminate Process:** The Behavior Analyzer analyses the system call log is to determine if an entry with system call ID corresponding to that of Terminate Process is logged. If such an entry is found, this behavior is logged in the behavior log for that process.

### 2. Creating, deleting or modifying files on the system:

Most of the malware behavior can be captured looking at the changes to the filesystem which involves creating, deleting or modifying files in some important locations. On a windows machine the malware mainly targets the system and windows folders.

**Create, read or modify files to system32 folder:** The system call log is analyzed to check if any of the processes creates, reads or modifies files with particular extensions in system32 directory. The

### Dynamic Malware Analysis on Feather Weight Virtual Machine

system32 directory is obtained by calling SHGetSpecialFolderPath on CSIDL\_SYSTEM. If any process creates files with extension “.dll”, “.exe”, “.sys”, “.bat”, “.ocx”, “.pif”, “.scr”, “.vbs” or “.ini”, the name of the file and the folder in which it was created is logged into the process’ log file.

Create, read or modify files to windows folder: The log is checked to see if any process creates, reads or modifies certain files in windows folder. The windows directory is obtained by calling SHGetSpecialFolderPath on CSIDL\_WINDOWS. If any of the files “system.ini”, “win.ini”, “wininit.ini” are modified/read or if any other files are created in windows directory or windows\Tasks directory, a log entry is made in the process’ log file to indicate the name of the file and the folder in which it was created. Figure 3.4 shows the content of such file created by FVM Analyzer.

Create, read or modify files to startup folder: The system call log is analyzed to check if any files are created, read or modified in startup folders. The startup directories are obtained by calling SHGetSpecialFolderPath on CSIDL\_COMMON\_STARTUP and CSIDL\_STARTUP. If any files were created in any of the two folders, a log entry is made to indicate the name of the file and the folder in which it was created.

```
*****  
Process ID: 1016, parent process ID: 948  
Process image name: C:\createfile.exe  
*****  
-- Created a.txt  
  in Windows Folder C:\WINNT\  
-- Created a.txt  
  in Windows Folder C:\WINNT\Tasks\  
*****
```

Figure 3.4 Content of Behavioral log for a process that creates files in the windows folder.

Create files in drive root: The system call log is checked to see if particular files are created in drive root. A check is performed to see if any of the files "explorer.exe", "autoexec.bat", "boot.ini", "config.sys", "ntldr", "ntdetect.com", "AutoRun.inf", "sos.exe", "System.exe" have been created in drive root. Also a check is done if files with extensions ".inf", ".msc" or ".url" have been created in drive root. If any of such files are created, a log entry is made in the process’ log file indicating the name (full path name) of the file that was created. This analysis helps in identifying processes that create an autorun file so that it executes whenever the drive is accessed

## Dynamic Malware Analysis on Feather Weight Virtual Machine

Modify host file: The log is analyzed to check if any write system call was made on a file in `system32\drivers\etc\hosts` directory. If any such entry is found, a log entry is created in the process' log file to indicate that it modified a file in `\system32\drivers\etc\hosts` directory.

### 3. Create, Modify or Delete registry entries:

On a Windows operating system, the Registry directory stores settings and options for the operating system software, hardware, users, and preferences for the system. Entries in this folder, called the registry entries are the main target for Malware since modifying a registry entry would change the settings or preferences for any software or system. The FVM Analyzer identifies any malicious changes to the registry entries by analyzing the system call log to check if any of the processes have created, modified or deleted particular registry entries. These checks make sure to identify any process that

- Marks some files mainly protected system files as hidden in an attempt to hide
- Modifies/creates registry entries so that task manager cannot be accessed
- Modifies registry entries so that it executes whenever Windows starts
- Creates registry entries to disable the registry tools and the folder options in Explorer
- Creates registry keys, which are used as infection markers
- Changes the start page of Internet Explorer
- Modifies registry entries to disable System Restore
- Modifies the registry entries so that it executes in safe mode
- Modifies registry entries so that the worm is executed whenever certain file types are opened
- Creates registry entries so that the folder is shared
- Modify `HKEY_CURRENT_USER\Software\Microsoft\Outlook Express`

4. Copy files: The system call log is analyzed to check if a process creates a section, reads a file and then writes to it in the same order and the output and input buffers in the Read and Write system calls match. If the name of the file being copied matches the image name of the process a copy self behavior is logged, else a copy of file behavior is logged.

5. Load Driver: Typical malware behavior also involves loading certain services/drivers to perform certain malicious tasks. The log is checked to see of any process loaded drivers that match the

## Dynamic Malware Analysis on Feather Weight Virtual Machine

configuration stored. If yes, a log entry is entered into the process' log file to indicate that the behavior.

6. Write to another process' Virtual Memory: The system call log is analyzed to check if there is a log entry for system call corresponding to NTWriteVirtualMemory. If there is an entry, it is logged into the process' log file that the process attempted to write to a different process' virtual memory.

7. Access Address Book: This behavior is identified by analyzing the log to check if the process has queried value key corresponding to the address book and then created a section in the same order. A check is also performed to see if the process has queried key and created section for the same file.

8. Connect to a network port: The system call log is scanned to check if there is any log entry with system call Id corresponding to WSPConnect. If an entry is found the behavior is logged.

## 4. LIMITATIONS AND FUTURE WORK

Even though FVM offers total abstraction of the OS to the applications running in the VM, it is not difficult for a process to detect that it is running in a virtualized environment. The maximum path name length that can be created on FVM is lesser than what windows supports, since all the files of the VM are created in a directory for the VM which is a subdirectory on the actual host machine. This behavior can be exploited by malware to detect virtual machines and hence change its behavior.

The FVM System call and API logging can further be enhanced by intercepting additional APIs to cover additional behavior. For example, WIN32 API related to enumerating Windows can be intercepted to detect malware that try to perform malicious actions on open windows. Certain other behavior pertaining to malware is still to be analyzed, for example the behavior of a process attempting to start the "shutdown" can be logged. Appendix A provides a list of high level malware behavior obtained by looking at the behavior of current malware threats [5]. Additional functionality can be added to the FVM Analyzer to identify all of the behavior listed.

Currently the FVM System Call logging and Analyzer detects any malicious behavior that would call any of the intercepted system calls or Network API. This Approach will fail if the intercepted API calls are not a part of the execution path of the malware. Some advanced malware implementations like Rootkits use API hooking and Direct Kernel Object Manipulation (DKOM). The malware program obtains privileged rights and hence, has direct access to kernel memory. This

## Dynamic Malware Analysis on Feather Weight Virtual Machine

allows it to achieve the goal of hooking APIs and hiding kernel objects like processes, files, and network connections.

Hooks can be detected on a system by looking at the SSDT and determine function pointers that do not resolve to `ntoskrnl` or any function pointers that belong to the FVM. This is not sufficient to effectively detect a Rootkit which uses DKOM. For example, on a Windows machine a Rootkit hides processes by modifying the active processes list in the Process Control Block, and removes the process to be hidden from the list. Removing the process from Active Processes list still doesn't affect scheduling since scheduling is based on threads. Hence the process continues to run, but is Hidden.

To identify such behavior, any hidden processes on the system are to be identified. One way is to look at the various threads on the scheduler queues and determine whether the corresponding process descriptor is correctly linked.

## 5. CONCLUSIONS

The System call and API logging modules and FVM Analyzer hence successfully log and analyze malicious behavior. From the above results and analyzed behavior it can be verified that the Feather weight Virtual Machine provides strict isolation for running malicious processes and provides a secure environment for their analysis without compromising the host environment.

## 6. REFERENCES

1. 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 ACM/USENIX Conference on Virtual Execution Environments (VEE'06). Ottawa, Canada, June 2006.
2. Yang Yu, "OS-level Virtualization and their Applications," P.H.D Dissertation, 2007.
3. Bayer Ulrich, Moser Andreas, Kruegel Christopher, Kirda Engin, "Dynamic Analysis of Malicious Code", Springer publications, May 2006.
4. Carsten Willems, Thorsten Holz, Felix Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox", IEEE Security and Privacy archive, Volume 5, Issue 2 (March 2007), Pages 32-39.
5. Latest risks and vulnerabilities listed at the Symantec website "The Threat Explorer" [http://www.symantec.com/business/security\\_response/threatexplorer/threats.jsp](http://www.symantec.com/business/security_response/threatexplorer/threats.jsp)

## APPENDIX A

List of High level behavior of Malware obtained from the technical details of threats listed at the Threat Explorer [5]

- Make copies of itself
- Mark files as a hidden and protected system files in an attempt to hide.
- Connect to random IP addresses
- Create registry keys, which are used as infection markers.
- Obtain email addresses by searching documents on the disk or by accessing the Address Book.
- Construct attachments using specific filenames and extensions and send emails.
- Enumerate all open windows and perform malicious actions.
- Change the start page of Internet Explorer.
- Steal user names and passwords.
- Create mutex so only one instance of the worm is running
- End the security-related services like MpfService, GuardDogEXE, SmcService, OutpostFirewall, vsmon AVGfwSrv
- Attempt to disable the Windows Firewall and the Windows Security Center
- Modify registry entries to disable System Restore
- Modify registry entries to execute in safe mode
- Modify registry entries so that the worm is executed whenever certain file types are opened
- Create autorun files in Drive root so that the worm executes whenever the drive is accessed
- Infect files by appending invisible IFRAME tag
- Create registry entries so that the folder is shared
- Hook the Windows authentication APIs and collects user information, such as login names and passwords, and may alter the behavior of the login shell
- Inject itself into Explorer.exe, so that it can log keystrokes typed on the compromised computer

## Dynamic Malware Analysis on Feather Weight Virtual Machine

- Attempt to run an ICMP Denial of Service (DoS) attack using the ping command against the remote hosts
- Attempt to restart the computer by issuing the following command `shutdown -r -t 0`
- Open an RTSP connection to a malicious server.

### APPENDIX B:

The System call log is analyzed to see if processes create, modify or delete entries in certain registries. The registry entries are in any of HKEY\_LOCAL\_MACHINE, HKEY\_CURRENT\_USER or HKEY\_USERS. The list of registry entries is given below:

```
"SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects",
"SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
"SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce",
"SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx",
"SOFTWARE\Microsoft\Internet Explorer\Toolbar",
"SOFTWARE\Microsoft\Internet Explorer\Search",
"SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify",
"SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows",
"SOFTWARE\Microsoft\Internet Explorer\Toolbar\WebBrowser",
"SYSTEM\CurrentControlSet\Control\Class\{4D36E96B-E325-11CE-BFC1-08002BE10318}",
"SOFTWARE\Microsoft\Windows\CurrentVersion\policies\Explorer",
"Software\Microsoft\Windows\CurrentVersion\Policies\System",
"SYSTEM\CurrentControlSet\Control\SafeBoot",
"SYSTEM\ControlSet001\Control\SafeBoot",
"SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options",
"SOFTWARE\Microsoft\Internet Explorer\Main",
"SOFTWARE\MICROSOFT\Windows\CURRENTVERSION\Internet Settings\Cache",
"SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug"
"Software\Microsoft\Windows NT\CurrentVersion\Winlogon",
"Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run",
"Software\Microsoft\Windows NT\CurrentVersion\WOW\boot",
"Software\Microsoft\Windows NT\CurrentVersion\IniFileMapping",
"Software\Microsoft\Windows\CurrentVersion\explorer\shell folders",
"Software\Microsoft\Windows\CurrentVersion\explorer\user shell folders",
```

## Dynamic Malware Analysis on Feather Weight Virtual Machine

"Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved",  
"Software\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad",  
"Software\Microsoft\Active Setup\Installed Components",  
"Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects",  
"Software\Microsoft\Outlook Express",  
"Software\Microsoft\Command Processor",  
"System\ControlSet???\Control\Session Manager\Environment",  
"System\ControlSet???\Control\Session Manager",  
"Software\Classes\?\shellex\contextmenuhandlers",  
"Software\Classes\exefile\shell\open\command",  
"Software\Classes\exefile\shell\runas\command",  
"Software\Classes\Folder\shell\open\command",  
"Software\Classes\Folder\shell\explore\command",  
"Software\Classes\batfile\shell\open\command",  
"Software\Classes\comfile\shell\open\command",  
"Software\Classes\cmdfile\shell\open\command",  
"Software\Classes\regfile\shell\open\command",  
"Software\Classes\scrfile\shell\open\command",  
"Software\Classes\scrfile\shell\config\command",  
"Software\Classes\vbsfile\shell\open\command",  
"Software\Classes\vbsfile\shell\open2\command",  
"Software\Classes\jsfile\shell\open\command",  
"Software\Classes\jarfile\shell\open\command",  
"Software\Classes\piffile\shell\open\command",  
"Software\Classes\htafile\shell\open\command",  
"Software\Classes\jarfile\shell\open\command",  
"SOFTWARE\Classes\txtfile\shell\open\command",  
"SOFTWARE\Classes\hlpfile\shell\open\command",  
"SOFTWARE\Classes\chm.file\shell\open\command",  
"Software\Classes\.exe",  
"Software\Classes\.Folder",  
"Software\Classes\.bat",  
"Software\Classes\.com",  
"Software\Classes\.cmd",  
"Software\Classes\.reg",  
"Software\Classes\.scr",

## Dynamic Malware Analysis on Feather Weight Virtual Machine

```
"Software\Classes\.vbs",
"Software\Classes\.js",
"Software\Classes\.jar",
"Software\Classes\.pif",
"Software\Classes\.hta",
"Software\Classes\mailto\shell\open\command",
"System\ControlSet???\Control\WOW",
"Environment",
"Control Panel\Desktop",
"System\ControlSet???\Services\Tcpip\Parameters",
"System\ControlSet???\Services\SharedAccess\Parameters\FirewallPolicy",
"SYSTEM\CurrentControlSet\Control\Lsa",
"S-1-5-21-1150637735-552541835-915212665-500\Software\Microsoft\Windows\CurrentVersion\Policies\System",
"S-1-5-21-1150637735-552541835-915212665-500\Software\Microsoft\Windows\CurrentVersion\Policies\WindowsUpdate",
"S-1-5-21-1150637735-552541835-915212665-500\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced",
"Software\Microsoft\Windows\CurrentVersion\Run\\"Yahoo\" = ",
"Software\Microsoft\Internet Explorer\Main\\"Start Page\" = ",
"Software\Microsoft\Internet Explorer\Main\\"Local Page\" = ",
"Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\\"Hidden\" = \"1\"",
"SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Taskmgr.exe",
```

The system call log is analyzed to see if any process creates, modifies or deletes files in drive root, windows folder (and some sub folders), system32 folder (and some subfolders) and startup folders. Typical paths for these folders and their subfolders that are checked for in the analyzer are given below:

```
C:\Windows
C:\Windows\Tasks
C:\Windows\system32
C:\Windows\system32\drivers\etc\hosts
C:\Documents and Settings\All Users\Start Menu\Programs\Startup
C:\Documents and Settings\username\Start Menu\Programs\Startup
```