ISSN: 2321-7782 (Online)
e-ISJN: A4372-3114
Impact Factor: 6.047

Volume 5, Issue 2, February 2017

International Journal of Advance Research in Computer Science and Management Studies

Research Article / Survey Paper / Case Study

Available online at: www.ijarcsms.com

On Demand Virus Scanning

J. Rajsekhar[1]
Assistant professor of CSE dept.
St. Peter`s engineering college,
Hyderabad – India

V. Harika Naidu[2]
Student of CSE dept.
St. Peter`s engineering college,
Hyderabad – India

V. Shreya[3]
Student of CSE dept.
St. Peter`s engineering college,
Hyderabad – India

Abstract: ***The biggest problems faced by the Linux user is protection from viruses that damage the directory files of the user's operating system. Root folder has all the directories of the system like /bin, /sbin, /etc etc. Viruses usually attack files of /bin which consists of system binary and if infected will crash or not let the system boot properly. Such problems can cause other types of infections to other root sub directories.***

All of the antivirus programs protect our system from viruses. But not each of them does the same work, as there are different approaches for each anti-virus.

Avast: It has a good Linux GUI , which is easy to use by the users but it does not block a program (ie virus) that is modifying the system files .It has a own virus database but does not contain a global virus database that has all the types of viruses.

Avg: It has a good scanning feature that easily detects viruses. Avg also has a strict approach i.e. it does not give any chance to the virus to infect the system, but the major drawback is that it occupies a lot of system resources i.e. ram, processor etc. And also it uses command line (i.e. it is not in a GUI) which is not user friendly.

When compared to the antiviruses available in the market, each has their own individual protection features/methods. Most of the antiviruses does not monitor the root directories and also have the global virus database present simultaneously in it.

The antivirus we are presenting consists monitoring the root directories, security events, system calls and commands run by the user. It also consists of multiple types of virus databases and does not take a lot of system resources.

Keywords: Malware; Anti-virus; Linux {root-directories}; Virus-detection.

I. INTRODUCTION

Linux operating systems are often considered to be immune to malware attacks, which would mean that antivirus software for Linux would be redundant. In reality, the situation is not so simple. Linux malware does exist, even if the number of programs is small; for example, in March 2014, ZDNet reported the discovery of the cybercrime campaign "Operation Windigo". One of Windigo's components – Ebury – provided attackers with a backdoor to infected servers and the ability to steal SSH credentials and send spam mails. Researchers observed that Ebury had infected approximately 26,000 Linux servers since May 2013.

Another reason for using an antimalware program on a Linux computer is to intercept any Windows malware before it can be passed on to a Windows system that it could infect.

Malware for Linux systems:

Most Linux-malware targets the server space, not desktops. Therefore, Anti-Virus software is mostly needed on Linux file and mail servers. Nevertheless, Linux desktops are not completely safe either, as there exists also cross-platform malware and phishing is a threat for any operating system. Furthermore, as mentioned previously, Linux users might receive and save (malicious) file attachments on their Linux machine and act as a vector for Windows malware.

One reason why the number of Linux malware programs is relatively small might be the large number of existing Linux kernel versions. Not only are there various different standard versions that are currently in use, some distributions also use a customized version of the Linux kernel.

A survey, which gathered data concerning the different Linux kernel versions used on Linux servers, showed that there were almost 1,300 different Linux kernel version distributed among the roughly 20,000 Linux servers included in the survey.

Not only the kernel itself, but also the software stack on top of it comes in the form of hundreds of different Linux distributions.

Linux security advice:

Employing good security practices can help to further secure your Linux system. We recommend the following:

1. Keep installed software up-to-date
2. Use phishing protection (at least the one provided in most browsers)
3. Only install software from trusted sources (e.g., the package manager of your Linux distribution)
4. Don't log on as root – use the sudo utility to gain temporary administrator access
5. Use strong passwords
6. Disable services that you don't use (IPv6, for instance)
7. Don't run commands you do not understand
8. Backup your data/system regularly

II. PRELIMINARY WORK

A. Virus Definitions

Antivirus software depends on the virus definitions to identify malware. That is the reason it updates on the new viruses definitions. Malware definitions contain signatures for any new viruses and other malware that has been classified as wild. If the antivirus software scans any application or file and if it finds the file infected by a malware that is similar to the malware in the malware definition. Then antivirus software terminates the file from executing pushing it to the quarantine. The malware is processed accordingly corresponding to the type of antivirus software.

It is really essential for all the antivirus companies to update the definitions with the latest malware to ensure PC protection combating even the latest form of malicious threat.

B. Ways to get rid of viruses

- Signature-based detection
- Heuristic-based detection
- Behavioural-based detection
- Sandbox detection

- Data mining techniques

Signature-based detection - This is most common in Traditional antivirus software that checks all the .EXE files and validates it with the known list of viruses and other types of malware. or it checks if the unknown executable files shows any misbehaviour as a sign of unknown viruses.

Files, programs and applications are basically scanned when they in use. Once an executable file is downloaded. It is scanned for any malware instantly. Antivirus software can also be used without the background on access scanning, but it is always advisable to use on access scanning because it is complex to remove malware once it infects your system

Heuristic-based detection - This type of detection is most commonly used in combination with signature-based detection. Heuristic technology is deployed in most of the **antivirus programs**. This helps the antivirus software to detect new or a variant or an altered version of malware, even in the absence of the latest virus definitions.

Antivirus programs use heuristics, by running susceptible programs or applications with suspicious code on it, within a runtime virtual environment. This keeps the vulnerable code from infecting the real world environment.

Behavioural-based detection - This type of detection is used in Intrusion Detection mechanism. This concentrates more in detecting the characteristics of the malware during execution. This mechanism detects malware only while the malware performs malware actions.

Sandbox detection - It functions most likely to that of behavioral based detection method. It executes any applications in the virtual environment to track what kind of actions it performs. Verifying the actions of the program that are logged in, the antivirus software can identify if the program is malicious or not.

Data mining techniques - This is of the latest trends in detecting a malware. With a set of program features, Data mining helps to find if the program is malicious or not.

Critical paths for your Linux: The file directory on your Linux contains a number of critical paths that house applications, language frameworks, and tools. Some paths are common across servers, while others differ between control panels and versions.

Below are the locations of important Linux binaries:

Perl: /usr/bin/perl

Php: /usr/bin/php

Sendmail: /usr/sbin/sendmail

Below are the locations of important control panel-dependent paths:

CPanel	
HTTP Document Root	/home/
HTTPS Document Root	/home/
CGI-BIN	/home/
HTTP Logs	/usr/local/apache/domlogs/
Simple Control Panel	
HTTP Document Root	/home/
HTTPS Document Root	/home/
CGI-BIN	/home/
HTTP Logs	/var/log/httpd/
Parallels Plesk Panel version 7.5.3 and below	
HTTP Document Root	/home/httpd/vhosts/
HTTPS Document Root	/home/httpd/vhosts/

CGI-BIN	/home/httpd/vhosts/
HTTP Logs	/home/httpd/vhosts/
Parallels Plesk Panel version 7.5.4 and above	
HTTP Document Root	/var/www/vhosts/
HTTPS Document Root	/var/www/vhosts/
CGI-BIN	/var/www/vhosts/
HTTP Logs	/var/www/vhosts/
No control panel	
HTTP Document Root	/etc/httpd/htdocs/
HTTPS Document Root	/etc/httpd/htdocs/
CGI-BIN	/etc/httpd/cgi-bin/
HTTP Logs	/etc/httpd/logs/

III. System Requirements

Hardware Requirements:

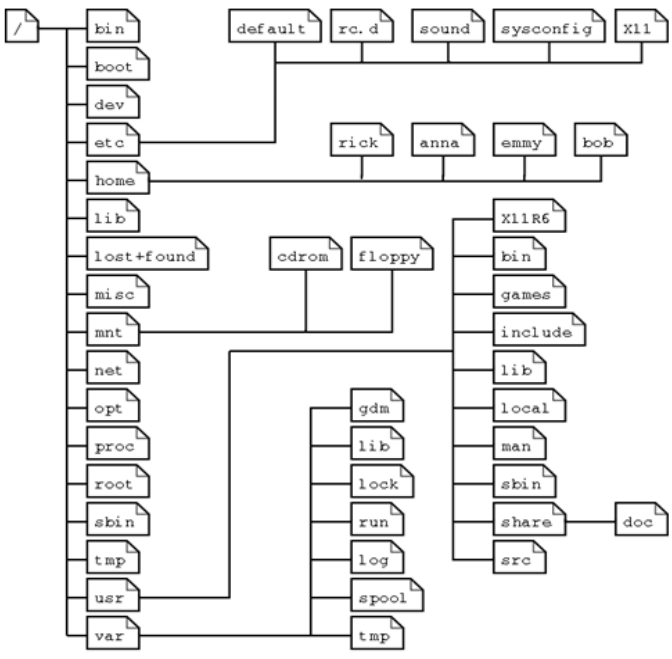
- RAM 1GB or Greater
- ROM min 4gb max up to user
- Processor more than 1.5 GHz

Software Requirements:

- Linux operating system (3.0 or Later)
- Inotify()
- Vim and gedit
- GCC compiler
- GDB

SYSTEM DESIGN

Root Directory Representation:



Algorithm:

Step 1: Start

Step 2: Boot the System

Step 3: Check if the directory is mentioned. If not go to Step 4 else skip Step 4.

Step 4: Ask the system/user for directory then go to Step 3.

Step 5: Check if any event is taking place. if not go to Step 6 else skip Step 6.

Step 6: Wait for the event to occur and go to Step 5.

Step 7: **D**isplay an alert message to the user.

Step 8: Check if log files are opened. If not go to Step 9 else skip Step 9.

Step 9: Create a new file and open it.

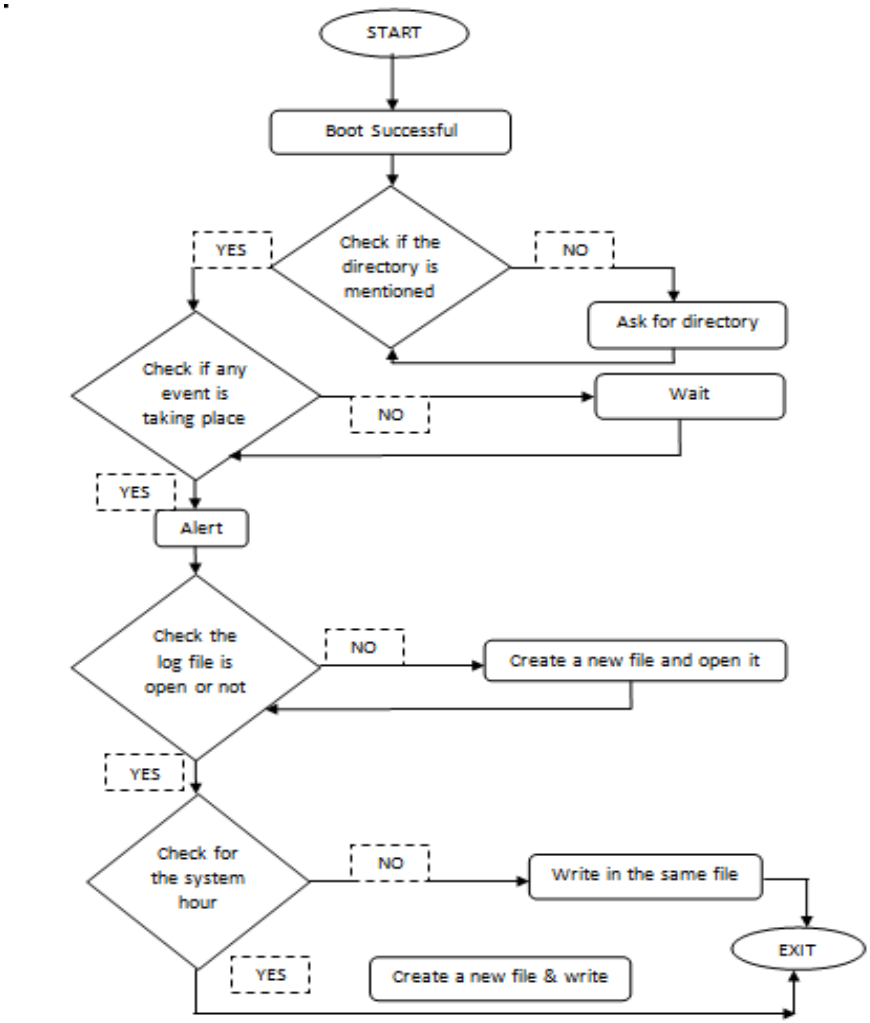
Step 10: Check for system time for creating different log files, if Hour passed go to Step 11 else skip Step 11.

Step 11: Write into the same file.

Step 12: Create a new file and write in it.

Step 13: End

FLOW CHART:



IV. IMPLEMENTATION

The directories that are mainly to be protected are bin and sbin. If these files are effected the booting process will be stopped.

/bin is a standard subdirectory of the *root directory* in Unix-like operating systems that contains the *executable* (i.e., ready to run) programs that must be available in order to attain minimal functionality for the purposes of *booting* (i.e., starting) and repairing a system.

The root directory, which is designated by a forward slash (/), is the *top-level* directory in the hierarchy of directories (also referred to as the *directory tree*) on Unix-like operating systems. That is, it is the directory that contains all other directories and their subdirectories as well as all files on the system.

A directory in a Unix-like operating system is merely a special type of *file* that contains a list of the names of *objects* (i.e., files, links and directories) that appear to the user to be in it along with the corresponding *inodes* for each object. A file is a named collection of related information that appears to the user as a single, contiguous block of data and that is retained in *storage* (e.g., a hard disk drive or a floppy disk). An inode is a *data structure* on a *filesystem* that stores all the information about a filesystem object except its name and its actual data. A data structure is a way of storing data so that it can be used efficiently. A filesystem is the hierarchy of directories that is used to organize files on a computer system.

The full names (also referred to as the *absolute pathnames*) of all of the subdirectories in the root directory begin with a forward slash, which shows their position in the filesystem hierarchy. In addition to /bin, some of the other standard subdirectories in the root directory include */boot*, */dev*, */etc*, */home*, */mnt*, */usr*, */proc* and */var*.

These are functions used for monitoring the events:

Inotify_add_watch: to manipulate the "watch list" associated with an inotify instance. Each item ("watch") in the watch list specifies the pathname of a file or directory, along with some set of events that the kernel should monitor for the file referred to by that pathname. Inotify_add_watch either creates a new watch item, or modifies an existing watch. Each watch has a unique "watch descriptor", an integer returned by inotify_add_watch when the watch is created.

SYNTAX: int inotify_add_watch (int fd, const char *pathname, uint32_t mask);

Structure of Inotify:

```
struct inotify_event {
   int      wd;       /* Watch descriptor */
   uint32_t mask;     /* Mask of events */
   uint32_t cookie;   /* Unique cookie associating related events (for rename(2)) */
   uint32_t len;      /* Size of name field */
   char     name[];   /* Optional null-terminated name */
};
```

Va_start Family Functions:

va_start(): The va_start () macro initializes ap for subsequent use by va_arg() and va_ends(), and must be called first.

The argument last is the name of the last argument before the variable argument list, that is, the last argument of which the calling function knows the type.

Because the address of this argument may be used in the va_start() macro, it should not be declared as a registered as a register variable, or as a function or an array type.

SYNTAX: void va_start(va_list ap, last);

va_arg(): the va_arg() macro expands to an expression to an expression that has the type and value of the next argument in the cal. The argument ap is the va_list ap initialized by va_start(0. Each call to va_arg() modifies ap so that the next call returns the next argument. The argument type is a type name specified type can be obtained simply by adding a * to type.

The first use of the va_arg() macro after that of the va_start(0 macro returns the argument after last. Successive invocations return the values of the remaining arguments.

If there is no next argument, or if type is not compatible with the type of the actual next argument(as promoted according to the default argument promotions(,random errors will occur.

If ap is passed to a function that uses va_arg(ap, type) then the value of ap is undefined after the return of that function

SYNTAX: type va_arg(va_list ap, type);

Va_end() : each invocation of a va_start() must be matched by a corresponding invocation of va_end() in the same function after they call va_end(ap) the variable ap is undefined. Multiple traversals of the list, each bracketed by va_start() and va_end are possible. va_end() may be a macro or a function.

SYNTAX: void va_end(va_list ap);

va_copy(): an obvious implementation would have a va_list be a pointer to the stack frame of the varied function. In such a setup (by far the most common) there seems nothing against an assignment

va_list aq = ap;

unfortunately, there are also systems that make it an array of pointers length ,and there one needs

va_list aq;

*aq = *ap;

Finally, on systems where arguments are passed in registers, it may be necessary for va_start() to allocate memory, store the arguments there, and also an indication of which argument is next, so that va_arg() can step through the list.

Each invocation of va_copy() must be matched by a corresponding invocation of va_end() in the same function. some systems that do not supply va_copy() have __va_copy insted, since that was the name used in the draft proposal

SYNTAX: void va_copy(va_list dest, va_list src);

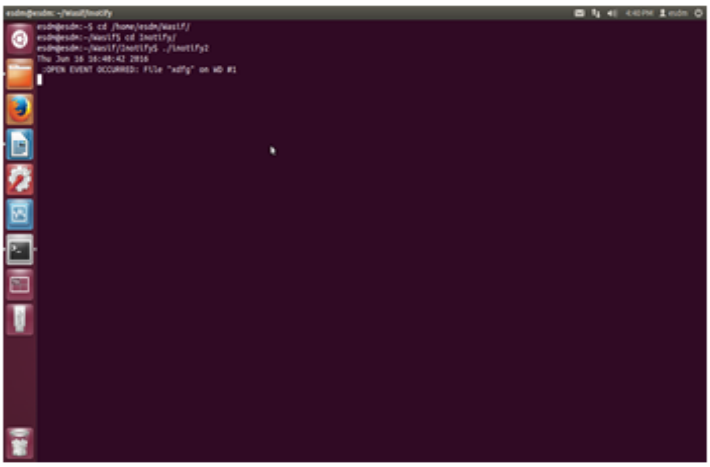
V. EVOLUTION

Test Case No.: 1

Input: Local Directory

Expected Outcome: All the events must be displayed.

Actual outcome:



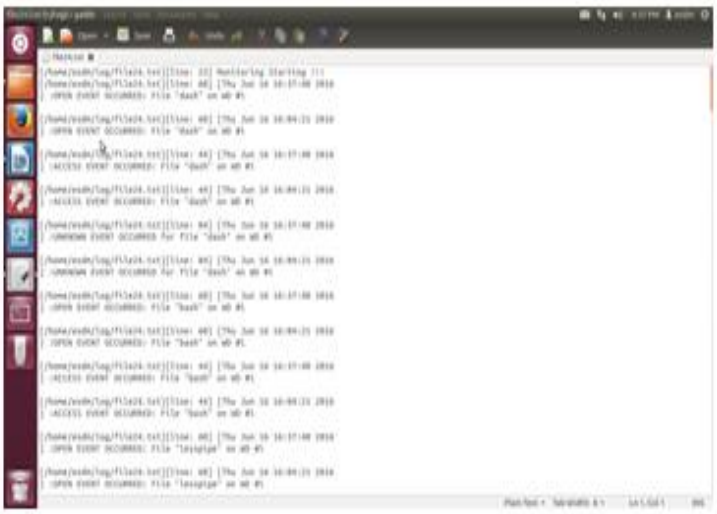
Result: Success.

Test Case No.: 2

Input: Local Directory to be saved in log files

Expected Outcome: must save all the details of the events until exit.

Actual outcome:



Result: Success.

VI. CONCLUSION

Computer viruses are not mortal they can be easily diminished, experimented by programmers. But we shouldn`t encourage a programmer who writes a code for virus that leads to destruction of something. If you do create a virus, make sure you know if it's working properly or else you might end up being in a troublesome situation. In order to deal with the viruses it is necessary to have a deep knowledge of the way in which different viruses exploits our systems weakness, thereby causing destruction of data or hampering of security. Furthermore, it is impossible to create antivirus against a particular virus without knowing the way it affects our system.

References

1. NETWORK BASED ANTI-VIRUS TECHNOLOGY FOR REAL-TIME SCANNING- HTTP://WWW.IJCSI.ORG/PAPERS/IJCSI-9-4-2-304-310.PDF
2. http://www.clamav.net/
3. Peter Szor, The Art of Computer Virus Research and Defence http://computervirus.uw.hu/index.
4. The new age of computer virus and their detection- http://airccse.org/journal/nsa/0512nsa05.pdf.
5. Study and comparison of virus detection techniques-https://www.ijarcsse.com/docs/papers/Volume_4/3_March2014/V4I3-0180.pdf.

