## A REPORT

**ON**

**ANTIVIRUS SOFTWARE BASED ON VIRUS**

**SIGNATURE METHOD**

###### BY

**P.RAJASHREE RAO - 2009A7PS012H**

UNDER THE SUPERVISION OF

**Dr. R.GURURAJ**

**Assistant Professor, Computer Science**

****

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**

**HYDERABAD CAMPUS**

**April,2012**

## A REPORT

**ON**

**ANTIVIRUS SOFTWARE BASED ON VIRUS**

**SIGNATURE METHOD**

Submitted in partial fulfillment of the

Computer Projects BITS C335

###### BY

**P.RAJASHREE RAO - 2009A7PS012H**

UNDER THE SUPERVISION OF

**Dr. R.GURURAJ**

**Assistant Professor, Computer Science**

****

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**

**HYDERABAD CAMPUS**

**April,2012**

**CERTIFICATE**

This is to certify that the report entitled, ----------------------------------------------------------------------------------------------------------------- and submitted by --------------------------------- ID No. ------------------ in partial fulfillment of the requirements of BITS C335 Computer Projects embodies the work done by him/her under my supervision.

Signature of the supervisor

Name:

Designation:

Date:

**ACKNOWLEDGEMENTS**

No project or research is done in a single day by a single person but it is a result of long experimentation and observation where the required environment consisting of many people play a vital role in initializing and completing a work.

First of all our heartfelt thanks to Dr.R.Gururaj, for constantly reviewing our project and resolving any difficulties we faced on our way. For giving us the initial guidance and encouragement regarding our project and insight regarding further research work possible in our area concerned.

We are indebted to BITS Pilani, for giving us the opportunity to experience project development.

Our words are incomplete without our acknowledgment towards our family and our friends, without whose encouragement and support we wouldn‟t have been able to complete our project.

**ABSTRACT**

With the ever expanding internet the probability of the malicious code entering the system is increasing. To avoid such code to harmfully affect the computer’s performance, detection of such code before they infect the system and also cleaning up of the already infected files is crucial to avoid further spread of such malicious codes.

The information can be scanned in two ways. One method involves comparing the information received with a virus database (known as 'virus signatures'). If the information matches any of the virus signatures, the antivirus concludes that the file is infected by a virus.

The other way of finding out if the information being scanned is dangerous, without knowing if it actually contains a virus or not, is the method known as 'heuristic scanning'. This method involves analyzing how the information acts and comparing it with a list of dangerous activity patterns.

In most cases tree structure is used in order to store virus signatures but scanning takes a lot of time as there are more number of memory access. Nowadays usage of bloom filters in association with good hash functions is becoming popular.

The antivirus software technology has to be updated now seeing the sheer volume of malwares flooding the systems.

**TABLE OF CONTENTS**

**1.What is a computer virus? 7**

**2.Types of viruses 8**

**3.How antivirus software works? 9**

**4.A generic virus scanner in java 12**

**5.Virus signature scanner using bloom filters and hash functions 15**

5.1 Design details of the scanner 16

5.2 A simple performance model 17

5.3 Scanner components 18

**6.Conclusion 21**

**7.References 23**

**1.WHAT IS A COMPUTER VIRUS?**

A computer virus is an **executable program**. Depending on the nature of a virus, it may cause damage of the hard disk contents, and/or interfere normal operation of a computer.

By definition, a virus program is able to replicate itself. This means that the virus multiplies on a computer by making copies of itself. This replication is intentional; it is part of the virus program. In most cases, if a file that contains virus is executed or copied onto another computer, then that computer will also be "infected" by the same virus.

A virus can be introduced to a computer system along with any software program. For Internet users, this threat can come from downloading files through FTP (file transfer protocol), or referencing email attachments.

When a virus is introduced to a computer system, it can attach itself to, or sometimes even replace, an existing program. Thus, when the user runs the program in question, the virus is also executed. This usually happens without the user being aware of it.

A virus program contains instructions to initiate some sort of "event" that affects the infected computer. Each virus has an unique event associated with it. These events and their effects can range from harmless to devastating. For examples:

* An annoying message appearing on the computer screen.
* Reduced memory or disk space.
* Modification of data.
* Files overwritten or damaged.
* Hard drive erased.

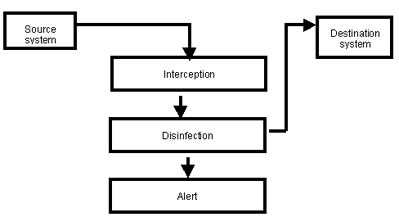
**2.TYPES OF VIRUSES**

There are many types of computer viruses:

* **File virus** : Most viruses fall into this category. A virus attaches itself to a file, usually a program file.
* **Boot sector virus** : These viruses infect floppy and hard drives. The virus program will load first, before the operating system.
* **Macro Virus** : This is a new type of virus that use an application's own macro programming feature to distribute themselves. Unlike other viruses, macro viruses do not infect programs; they infect documents.
* **Virus Hoax** : Although there are thousands of viruses discovered each year, there are still some that only exist in the imaginations of the public and the press - known as virus hoaxes. These viruses hoaxes do not exist, despite rumor of their creation and distribution.

**3.HOW ANTIVIRUS SOFTWARE WORKS?**

From the early viruses, created as experiments in the eighties, to the latest malicious code, one of the biggest worries for all computer users is the threat of viruses entering their systems.  
 An antivirus program is no more than a system for analyzing information and then, if it finds that something is infected, it disinfects it. The information is analyzed (or scanned) in different ways depending on where it comes from. An antivirus will operate differently when monitoring floppy disk operations than when monitoring e-mail traffic or movements over a LAN. The principal is the same but there are subtle differences.



The information is in the 'Source system' and must reach the 'Destination system'. The source system could be a floppy disk and the destination system could be the hard disk of a computer. The information interpretation system varies depending on whether it is implemented in operating systems, in applications or whether special mechanisms are needed.

The interpretation mechanism must be specific to each operating system or component in which the antivirus is going to be implemented. For example, in Windows 9x, a virtual driver VxD is used, which continually monitors disk activity. In this way, every time the information on a disk or floppy disk is accessed, the antivirus will intercept the read and write calls to the disk, and scan the information to be read or saved.

Antivirus products that are not specially designed for operating systems, but are implemented over other applications, have a different interpretation mechanism. For example, in an antivirus for CVP Firewalls, it is the firewall that provides the antivirus with information in order to scan it through the CVP protocol and in the antivirus for SendMail, the MilterAPI filter

facilitates information interpretation.

Once the information has been scanned, using either method, if a threat has been detected, two operations are performed:

**1**.The cleaned information is returned to the interpretation mechanism, which in turn will return it to the system so that it can continue towards its final destination. This means that if an e-mail message was being received, the message will be let through to the mailbox, or if a file way

being copied, the copy process will be allowed to finish.  
**2.** A warning is sent to the user interface. This user interface can vary greatly. In an antivirus for workstations, a message can be displayed on screen, but in server solutions the alert could be sent as an e-mail message, an internal network message, an entry in an activity report or as some kind of message to the antivirus management tool.

**Virus scan engines:**

Regardless of how the information to be scanned is obtained, the most important function of the antivirus now comes into play: the virus scan engine. This engine scans the information it has intercepted for viruses, and if viruses are detected, it disinfects them.

The information can be scanned in two ways. One method involves comparing the information received with a virus database (known as 'virus signatures'). If the information matches any of the virus signatures, the antivirus concludes that the file is infected by a virus.

The other way of finding out if the information being scanned is dangerous, without knowing if it actually contains a virus or not, is the method known as 'heuristic scanning'. This method involves analyzing how the information acts and comparing it with a list of dangerous activity patterns.

For example, if a file that can format a hard disk is detected, the antivirus will warn the user. Although it may be a new formatting system that the user is installing on the computer rather than a virus; the action is dangerous. Once the antivirus has sounded the alarm, it is up to

the user whether the danger should be eliminated or not.

Both of these methods have their pros and cons. If only the virus signatures system is used, it is important to update it at least once a day. As 15 new viruses are discovered everyday, an antivirus that is left for two or three days without being updated is a serious danger.

The heuristic system has the drawback that it can warn about items that are not viruses. If we have to work with a lot of items that may be considered dangerous, we could soon tire of the alerts. Programmers in particular may prefer to disable this option.

**4.A GENERIC VIRUS SCANNER IN JAVA**

Here is a brief overview of the scanning tool which gives a description of the main java classes from which the program is structured. Java , being an object oriented language, permits data encapsulation and abstraction, and these features can be used in partitioning data and functions. As a result, future additions and modifications to this program will not change the overall organization of data drastically.

1. T.java

Contains:

int list;

char hex;

list: stores the serial number of the virus signature in the database so as to provide whole information once the particular virus is detected.

hex: stores one of the hexadecimal characters that are the building blocks of the virus signatures.

2. Node1.java:

Contains:

public T data;

public List children;

data: Type Generic of class T

Represents each node of the tree (built from virus signatures)

children: Type list

Stores the children nodes for each node ‘data’.

3. viruscop.java

Contains:

String serialnumber;

String virusname;

String virussignature;

serialnumber: The serial number of the virus signature from the database

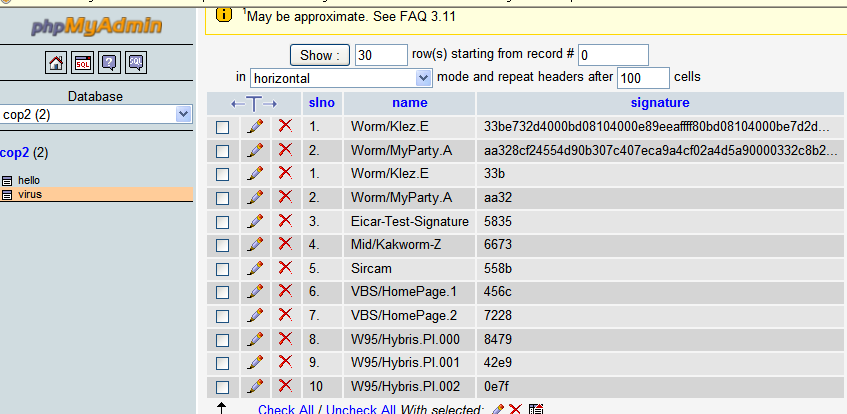
virusname: It stores the name of the virus taken from the database.

Virussignature: It stores signature of the virus after reading from the database.

4. database.java

Contains:

Connection to the database. The name of the database is cop2 having a table virus that contain the serial number, virus name and virus signature.



An object for each virus is created and stored in a list of type ‘viruscop’.

5. Input1.java

Contains:

A hex character is taken from a binary executable file that has to be scanned and sent to ‘search’ function of ‘genecop1.java’

6. genecop1.java

Contains:

a)construct function: This function constructs tree data structure based on the virus signatures present in the database. One virus signature is taken from the database each time and added to the tree. In this way tree structure is optimized.

b)traversal function: Postorder traversal of virus tree is implemented in order to check structure of tree.

c)search function: to check if the virus is present in the input file or not . Each character is taken from the input file and is passed down along the tree. If any node on the path of traversal contains a serial number other than zero then virus is detected. If no node is found corresponding to the hexadecimal character in the tree then once we start traversing from the root of the tree.

**5.VIRUS SIGNATURE SCANNING USING BLOOM FILTERS AND HASH FUNCTIONS**

In the age of Internet and the Web, viruses proliferate and spread easily. As a result, anti-virus technologies are a must in today’s wired world. An effective defense needs virus-scanning performed at every major network traffic stop and at the end-host computers.

Virtually all virus-scanning programs spend the bulk of their time matching data streams with a set of known virus signatures, and they all utilize some form of multi-pattern string matching algorithm. The number of virus signatures today is over 100,000 and is growing constantly. Unlike intrusion detection signatures, virus signatures cannot be neatly separated into rule sets consisting of a small number of strings. As a result, data structures used by these algorithms cannot fit in the CPU cache, and instead reside in main memory. Traditional matching algorithms require at least one random memory access per scanned byte.

A new virus scanning technique has been proposed that aims to ride the CPU speed curve. Here combination of a set of hash functions and a bloom filter array that fits in the CPU second-level (L2) cache constitute the virus signature scanner. We can determine the majority of the “no match” cases quickly, without incurring main memory accesses. Since the majority of network traffic does not contain viruses, most of the data stream belongs to the “no-match” case. The principle behind this scanner is the following. Currently, one L2 cache miss costs about 150-200 CPU cycles, and the speed gap keeps increasing. While reading new incoming data, the CPU has to take a cache miss. Once a piece of data has been brought from main memory into the CPU cache, this scanner can scan the data using its cache-resident bloom filter while the CPU waits for the next cache miss.

**5.1 DESIGN DETAILS OF THE SCANNER**

Main assumption while implementing this scanner is that, the vast majority of the data do not contain viruses. Therefore, it aims to determine the no-match cases with *high accuracy*, *minimal main-memory access* and *a small number of CPU instructions*. It achieves the goals by

using a filter that fits in CPU caches and acts as a first-pass scan to determine if the data need to go through an exact match algorithm.

Specifically, a sliding window of ‘w’ bytes slides down the input stream. For each ‘w’ bytes under the window, k hash functions are applied to calculate their hashes. The hash results are then used to probe into a bit array of N bits, which is a bloom filter pre-constructed from the virus signatures.

Scanner constructs a bloom filter from the set of plaintext signatures. The bloom filter is a vector of N bits, initially all set to 0. For each plain-text signature, k hash functions are applied to its first ‘w’ bytes , with results h1(a), h2(a),……, hk(a), all in the range of 1,….N. The bits at positions h1(a),h2(a),………, hk(a) are then set to 1. At scanning time, it moves over the input data stream one byte at a time. For each ‘w’ byte block b, the scanning algorithm applies the first hash function, h1(b), and checks the corresponding bit in the bloom filter. If the bit is 1, it computes the next hash function h2(b); if not, it immediately goes over to the next byte, and starts applying hash functions over the next ‘w’-byte block.

In the case where all k functions have positive bloom filter matches, scanner needs to check for exact match. We need to pre-construct a “secondary hash table” using the last hash function hk, with each entry holding a linked list of signatures which are checked linearly.

**5.2 ASIMPLE PERFORMANCE MODEL**

Based on our prior experimentation in using bloom filters, k = 4 works well. Therefore, there are three choices left in setting up Hash-AV:

\_ Choosing four hash functions

\_ Choosing the size of the bloom filter

\_ Choosing ‘w’

We will assume that the four hash functions are h1, h2, h3, and h4, applied in that order. Furthermore, assume that the function hi can be calculated at ci MB/s. Let the total number of signatures be M, and the size of the bloom filter be M \* K bits. The function h1 then has a false positive probability of p1 in the bloom filter. The probability p1 is determined by both the hash

function and the bloom filter’s expansion factor K. Similarly, h2 has a false positive probability of p2,1 in h1’s false positive cases. In other words, p2,1 is the conditional probability of false positive under h2 given that h1 has a false positive. The ratios p3,2,1 and p4,3,2,1 are defined similarly. The performance of the scanning algorithm can be modeled using the above parameters. Note that h2 is called when h1 hits in the bloom filter (i.e. h1’s bit is 1), h3 is called when both h1 and h2 hit in the bloom filter, and h4 is called when all three previous hash functions hit in the bloom filter. Thus, the throughput of the scanning algorithm is:

c1 + p1\*c2 + p1\*p2,1\*c3 + p1\*p2,1\*p3,2,1\*c4 + p1\*p2,1\*p3,2,1\*p4,3,2,1\*C

where C is the cost of the exact string matching algorithm. Clearly, since all the probabilities are between 0 and 1, the hash functions should be ordered from the cheapest (computationally) to the most expensive.

Scanner uses “bad but cheap” hash functions as initial hashes, and relying on serial hash lookups. The “good but expensive” hash functions are only calculated when the cheap initial hashes indicate a match, effectively reducing the CPU computation of the bloom filter probe.

**5.3 SCANNER COMPONENTS**

To actually construct a scanner, we have to determine the variables values ‘w’,hash functions and K. Below, we use experiments to determine each component of the filter. Since

the choices are intertwined, we first fix ‘w’ to 7, and study the hash functions and bloom filter sizes. We then return to the choice of ‘w’ near the end.

***A. Selecting Hash Functions***

The criteria for the hash functions are that they should be cheap and they should produce relatively random distributions. We first chose a set of well-known fast hash functions from the open source community. The functions usually have 0.2% to 1% collision rate on our sample files, and work well on inputs longer than 4 bytes. Table 1 list the hash functions, giving their performance measurements over a sample executable of 120 MB, and the percentage of false positives in the filter.

For these tests, ‘w’ is 7 and the bloom filter size is 256 KB. The throughput measurement contains the cost of hashing each block and the overhead of probing the bloom filter to see if there is a match.

Hash Name Hash Perf % of unfiltered

(MB/s) input

fnv-32-prime 27.44 1.75%

djb2 43.79 1.77%

hashlib fast-hash 46.22 1.76%

sdbm 36.25 1.75%

ElfHash 25.30 7.46%

These hash functions are not considered good performance wise. They might be considered fast compared to other hash functions, but not compared to memory-copy speed, which goes at 260MB/s on the desktops.

Most scanners use two really fast “hash” functions: “mask” and “xor+shift”.

“Mask” takes the first four bytes, casts them to an integer, and chooses the lowest log2(N) bits, where N is the size of the bloom filter.

“Xor+shift” takes the first six bytes, casts bytes 0-3 into an integer, and xors this word with 0 to get the first hash value. It then repeats the same operation two more times, for bytes 1-4 and 2-5, always xoring with the previous hash value to get the next one. It then picks the lower log2(N) bits of the final integer to check against the filter. “Mask” and “xor+shift” can be computed at throughputs of 160 MB/s and 120 MB/s consecutively.

On virus signatures, “mask” and “xor+shift” can filter away 88% and 96% of the input bytes. As standalone hash functions their false positives would be too high. However, used as *first* *level hash functions*, they can effectively cut down the number of times that the “good” hash functions are calculated by an order of magnitude. Hence, Hash-AV contains the following four hash functions:

mask, xor+shift, fast hash from hashlib.c and sdbm .

***B. Selecting Bloom Filter Sizes***

Traditional bloom filter implementations choose filter sizes such that half of the filter bits are 1. In any scanner, however, a number of factors impact the choice of the bloom filter size:

1. the CPU cache effect: the portion of the filter that fits in the CPU cache, and the cache miss ratio in cases when the filter cannot all fit in the cache.

2. the initial hash function effect: the initial hash functions are much faster than the latter ones. However, how much of the input data that the initial hashes filter away depends

on the sparsity of the bloom filter.

3. the false hit ratio: A 3% false hit ratio in a bloom filtervmight be acceptable if the cost of the false hit is only an order of magnitude higher than the cost of a filter probe.

However, it would not be acceptable if the cost of the false hit is two orders of magnitude higher.

Clearly, the choices are intertwined, and depend on the relative ratio of the cache size and the size of the filter.

***C. Selecting* ‘w’**

The choice of ‘w’ is mainly affected by the distribution of signature lengths in the signature database. Generally, larger ‘w’s are preferred since strings that match the first ‘w’ bytes in a signature are more likely to match the actual signature.

On the other hand, dramatically increasing ‘w’ has two sideeffects. First, the hash functions take more time to compute the result, which in turn slows the algorithm down. Second, with a bigger ‘w’, Hash-AV has to leave out short signatures. The short signatures are then handled together with the polymorphic ones in a separate scan using the Aho-Corasick algorithm. The “xor+shift” function is designed to operate on six bytes of data, and other hash functions can’t distinguish input accurately for small input streams. Therefore, a lower limit of six is set on ‘w’.

It is seen that, through judicious use of the CPU cache, this scanner can sgnificantly improve the performance of the open-source virus scanner Clam-AV. By using cache-resident bloom filters, Hash-AV determines the vast majority of the “no-match” cases with no main memory accesses. By using cheap hash functions whose computational costs are hidden by memory access delays, it can potentially scan inputs at a third of memory copy speeds. Since the speed gap between CPU computations and random memory accesses continues to increase, we expect it to become more critical for virus scanning performance.

Key to scanners using bloom filter success is the use of very cheap functions such as “mask” and “xor+shift”. While in ordinary circumstances they are not good hash functions, they are very effective as initial functions in a serial application of a set of hash functions.

**6.CONCLUSION**

Although the anti-virus software attempts to update itself and overcome the malware threats, however one has to accept that virus authors are one step more ahead, because they decide how to attack first and anti-virus technologies have to only defense against their attacks.

Therefore, computer virology area needs more researches and investigation to be able to guess the future coming threats. There are many weaknesses in both viruses and anti-virus technologies, which must be studied and known well. Viruses usually look for the Achilles' heels in the defense system and attempt to attack them. Some major problems in detection methods are:

1 Most of detection methods are not powerful against evolutionary advanced or new viruses

2 Scanning process usually takes a considerable amount of time to search a system or networks for the patterns.

3 An anti-virus and its virus database need to be updated continually and extremely, otherwise it cannot be reliable.

The rapidly multiplying malware epidemic means the "static, old school," approach of virus signature based scanning technique is reaching the end of its useful life. The following are some of the newer techniques that are coming up:

* [Behavior](http://www.computerworlduk.com/news/security/16681/reputation-based-antivirus-software-comes-of-age/?intcmp=in_article;related) based protection
* [heuristics](http://www.computerworlduk.com/news/security/15577/symantecs-norton-anti-virus-2010-betas-go-live/?intcmp=in_article;related) and reputation analysis
* White-listing and blacklisting techniques

Another complication is that malware is now so artfully designed, "it spreads to 20 or 30 machines before it mutates, That means the neighbor has one variant and you have another. Hence the effectiveness of each signature has gone down.

Also, the cloud has to become real-time with sophisticated updating methods to be updated almost instantly. Web-threat protection is another arrow in the quiver to detect malware,

An example of this shift is seen at McAfee. Dave Marcus, director of security research and communications at McAfee, agrees signature-based detection "is less important than it was five years ago, when you consider the sheer volume of malware out there." McAfee has already begun a shift to cloud-based malware detection, and sees behavior-based detection as a good augmentation as well. But Marcus adds: "Signature-based recognition will always be part of security technologies going forward."

Hence even though the virus signatures might not totally be removed from the security checks but their efficiency is going down until and unless newer technologies are not used along with them. Thus this technology has to be updated now seeing the sheer volume of malwares flooding the systems.

**7.REFERENCES**

1. <http://www.ust.hk/itsc/antivirus/general>
2. [www.antivirusworld.com](http://www.antivirusworld.com)
3. en.wikipedia.org/wiki/Computer\_**virus**
4. en.wikipedia.org/wiki/bloom\_filters
5. crypto.stanford.edu/~cao/**hash**-**av**.pdf
6. [http://www.computerworlduk.com](http://www.computerworlduk.com/)