IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661, ISBN: 2278-8727 Volume 5, Issue 2 (Sep-Oct. 2012), PP 18-24
www.iosrjournals.org

Modeling and Containment of Uniform Scanning Worms

Namratha M[1], Pradeep[2]
[1, 2] *(Dept. of information science, PESIT/visvesvaraya technological university, India)*

Abstract: *Self-propagating codes called worms such as Code Red, Nimda, and Slammer have drawn significant attention due to their enormously adverse impact on the Internet. Thus, there is great interest in the research community in modeling the spread of worms and in providing adequate defense mechanisms against them. In this model, we present a (stochastic) branching process model for characterizing the propagation of Internet worms. The model is developed for uniform scanning worms and then extended to preference scanning worms. This model leads to the development of an automatic worm containment strategy that prevents the spread of a worm beyond its early stage. Specifically, for uniform scanning worms, we are able to determine whether the worm spread will eventually stop. We then extend our results to contain uniform scanning worms. Our automatic worm containment schemes effectively contain both uniform scanning worms and it is validated through simulations .*
The Internet has become critically important to the financial viability of the national and the global economy. Meanwhile, we are witnessing an upsurge in the incidents of malicious code in the form of computer viruses and worms. One class of such malicious code, known as random scanning worms, spreads itself without human intervention by using a scanning strategy to find vulnerable hosts to infect

Keywords*: Branching process model, Broadcast, Containment, Intrahost spreading, Scanning, Socket Communication, Worm*

I. Introduction

A computer worm[1] is a self-replicating computer program. It uses a network to send copies of itself to other nodes and it may do so without any user intervention. The name **worm** comes from *The Shockwave Rider*, a science fiction novel published in 1975 by John Brunner.

A computer virus[2] is a computer program that can copy itself and infect a computer in some form of executable code. Spreads by human action, sent over a network or the Internet. carried it on a removable medium such as a floppy disk, CD, DVD, or USB drive. Unlike a virus, it does not need to attach itself to an existing program. Worms almost always cause harm to the network, if only by consuming bandwidth, whereas viruses almost always corrupt or modify files on a targeted computer.

Worms[3] spread by exploiting vulnerabilities in operating systems. All vendors supply regular security updates and if these are installed to a machine then the majority of worms are unable to spread to it. If a vendor acknowledges a vulnerability but has yet to release a security update to patch it, a zero day exploit is possible. However, these are relatively rare. Users need to be wary of opening unexpected email, and should not run attached files or programs, or visit web sites that are linked to such emails. However, as with the ILOVEYOU worm, and with the increased growth and efficiency of phishing attacks, it remains possible to trick the end-user into running a malicious code. Anti-virus and anti-spyware software are helpful, but must be kept up-to-date with new pattern files at least every few days. The use of a firewall is also recommended. Computer worms -- malicious, self propagating programs -- represent a substantial threat to large networks. Since these threats can propagate more rapidly than human response [, automated defenses are critical for detecting and responding to infections. One of the key defenses against scanning worms which spread throughout an enterprise is *containment.*. Worm containment, also known as virus throttling, works by detecting that a worm is operating in the network and then blocking the infected machines from contacting further hosts. Currently, such containment mechanisms only work against *scanning* worms because they leverage the anomaly of a local host attempting to connect to multiple other hosts as the means of detecting an infectee. Within an enterprise, containment operates by breaking the network into many small pieces, or *cells*. Within each cell (which might encompass just a single machine), a worm can spread unimpeded. But between cells, containment attempts to limit further infections by blocking outgoing connections from infected cells.

A key problem in containment[4] of scanning worms is efficiently detecting and suppressing the scanning. Since containment *blocks* suspicious machines, it is critical that the false positive rate be very low. Additionally, since a successful infection could potentially subvert any software protections put on the host machine, containment is best effected inside the network rather than on the end-hosts. We have developed a scan detection and containment algorithm using java platform. The make our algorithm suitable for both hardware and software implementation. Evaluating the algorithm on traces from a large (6,000 host) enterprise, we find that with a total memory usage of 5 MB we obtain good detection precision while staying within a processing budget of at most 4 memory accesses (to two independent banks) per packet.

Worm containment systems have an epidemic threshold: if the number of vulnerable machines is few enough relative to a particular containment deployment, then containment will almost completely stop the worm. However, if there are more vulnerable machines, then the worm will still spread exponentially.

Finally, in general malicious attacks on worm containment systems[5],[6],[7]: what is necessary for an attacker to create either false negatives (a worm which evades detection) or false positives (triggering a response when a worm did not exist), assessing this for general worm containment, cooperative containment, and our particular proposed system. We specifically designed our system to resist some of these attacks. Worm containment is designed to halt the spread of a worm in an enterprise by detecting infected machines and preventing them from contacting further systems. Current approaches to containment are based on detecting the scanning activity associated with scanning worms, as is our new algorithm. The key component for today's containment techniques is *scanning:* responding to detected *portscans* by blocking future scanning attempts. Portscans--probe attempts to determine if a service is operating at a target IP address--are used by both human attackers and worms to discover new victims. Portscans have two basic types: *horizontal* scans, which search for an identical service on a large number of machines, and *vertical* scans, which examine an individual machine to discover all running services. The goal of scan suppression is often expressed in terms of preventing scans coming from “outside” inbound to the “inside”. If “outside” is defined as the external Internet, scan suppression can thwart naive attackers. But it can't prevent infection from external worms because during the early portion of a worm outbreak an inbound-scan detector may only observe a few (perhaps only single) scans from any individual source. Thus, unless the suppression device halts all new activity on the target port it will be unable to decide, based on a single request from a previously unseen source, whether that request is benign or an infection attempt. For worm *containment,* however, we turn the scan suppressor around: ``inside" becomes the enterprise's larger internal network, to be protected from the ``outside" local area network. Now any scanning worm will be quickly detected and stopped, because (nearly) *all* of the infector’s traffic will be seen by the detector.

II. Design Considerations

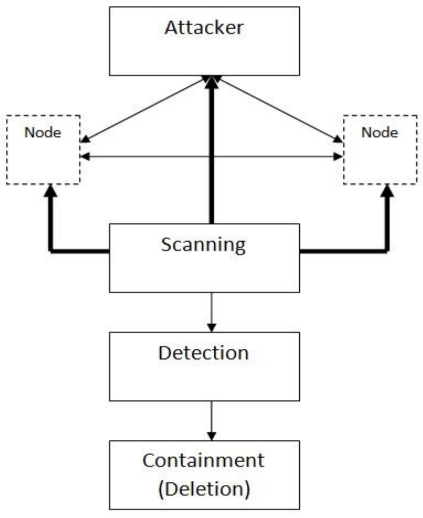


Fig 1: High level design

Functionality:
Step 1: The attacker sends the worm to each of the other host
Step 2: Each host intern sends it to each of the other host
Step 3: This process continues until stop spreading is pressed.
Step 4: Simultaneously in each host, scan the specified path.
Step 5: Delete all the detected worm files.

2.1 Branching Process Model[8]:

To the problem of combating worms[9], we have developed an inter and intra branching process model to characterize the propagation of Internet worms. Unlike deterministic epidemic models present earlier, this model allows us to characterize the early phase of worm propagation.

Inter host spreading: This module is responsible for modeling the spread of uniform scanning worms over a network, i.e., from one host to another.

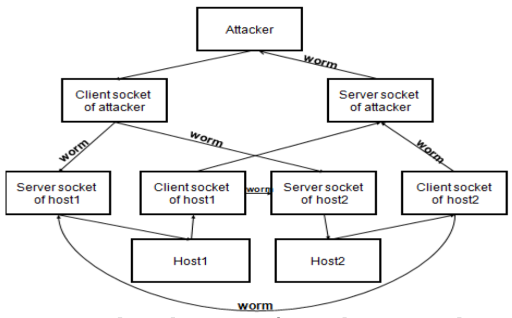


Fig 2: Flow diagram of inter host spreading

Functionality:
The sequence of steps is as follows:
Step 1: Create server socket for each of the host.
Step 2: On the primary attacker select the type of the worm to spread in the network.
Step 3: Create client socket on the primary attacker node
Step 4: Send the worm through the client socket to listening server socket of each of the other nodes in the network
Step 5: Now each node sends the worm through its client socket to the server socket of the other node.
Step 6: repeat step 5 until stop spreading is pressed.
Intra host spreading: This module is responsible for modeling the spread of worms[10] through a replication mechanism within a node in the network.

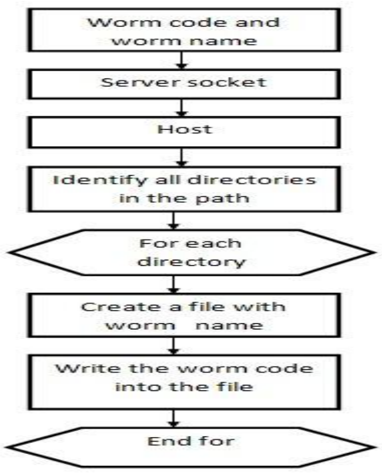

Worm code and worm name
Server socket
Host
Identify all directories in the path
For each directory
Create a file with worm name
Write the worm code into the file
End for

Fig 3: Flow diagram of intrahost spreading

2.2 Scanning:

Our strategy is based on limiting the number of scans to the specified path in the system. Our worm containment schemes effectively contain uniform scanning worms and it is validated through simulations and real trace data to be non-intrusive**.** This module is responsible for scanning the specified path in the system for the presence of the specific worm.

```
Step 1: Start.
Step 2: t1:=start time
 Step 3:=for each directory
Step 4: dir_count++
Step 5: for each directory in the file
 Step 6: file_count++
Step 7: Display current_file being scanned
Step 8:  if file_name != worm_name then goto step 12.
Step 9: detections_count++
Step 10: Worm containment
Step 11: Display path of detected worm file.
Step 12: End for
Step 13: End for
Step 14: t2=stop time
Step 15: t:= t2-t1
Step 16: Display detections_count, dir_count, file_count, time_to_scan;
Step 17: Stop
```

2.3 Containment of worms:

This module leads to the development of an automatic worm containment strategy that prevents the spread of a worm, specifically for uniform scanning worms. This module is responsible for the deletion of the detected files.

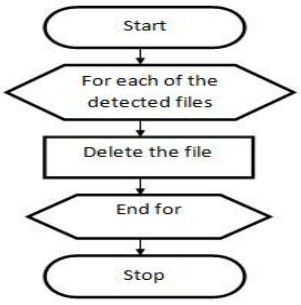


Fig 4: Flow diagram for containment of worms

2.4 Client-Server Framework:

The client-server framework[11] has been used for the modeling of worm spread across a network. The client sends the worm and the server receives the worm. Hence the client is associated with the attacker and the server is associated with the node being attacked in the network. The client and server communicate with each other by creating sockets at specific port numbers and creating Input/Output DataStreams to pass request messages and responses to enable communication.

III. Experiments And Results

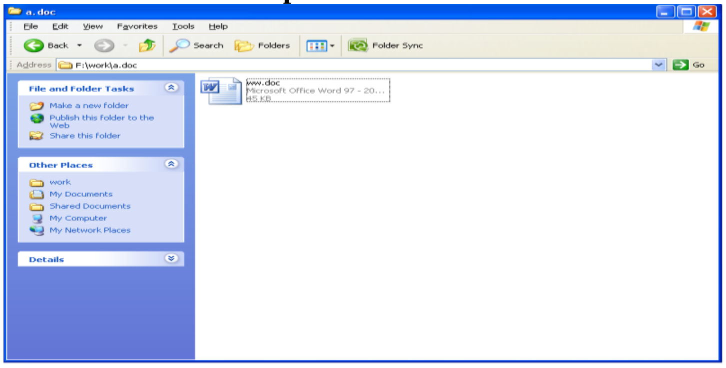
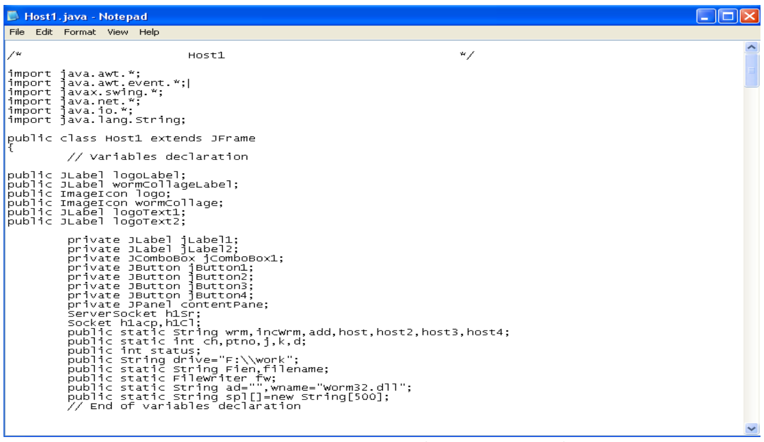
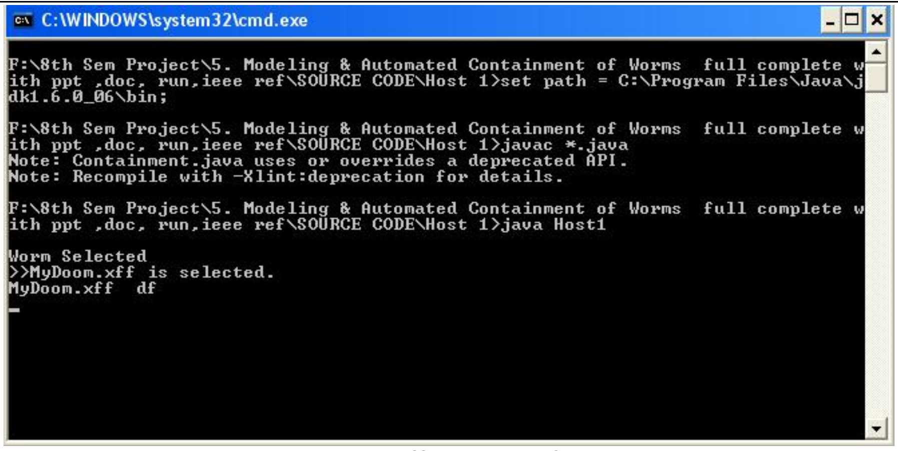


Fig 5: Before spreading there is no worm file in the folder a.doc



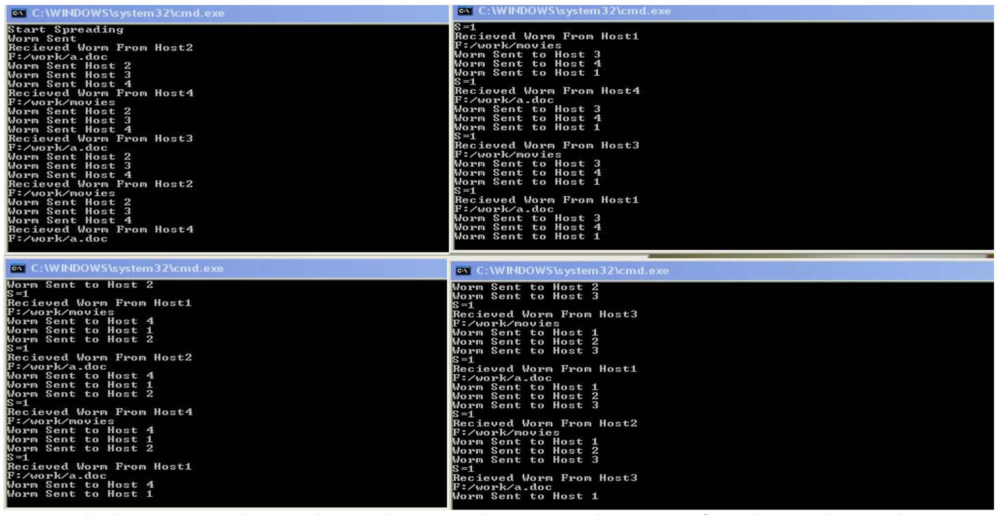
```
/*                          Host1                              */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;
import java.io.*;
import java.lang.String;

public class Host1 extends JFrame
{
        // Variables declaration

public JLabel logoLabel;
public JLabel wormCollageLabel;
public ImageIcon logo;
public ImageIcon wormCollage;
public JLabel logoText1;
public JLabel logoText2;

        private JLabel jLabel1;
        private JLabel jLabel2;
        private JComboBox jComboBox1;
        private JButton jButton1;
        private JButton jButton2;
        private JButton jButton3;
        private JButton jButton4;
        private JPanel contentPane;
        ServerSocket h1Sr;
        Socket h1acp,h1Cl;
        public static String wrm,incWrm,add,host,host2,host3,host4;
        public static int ch,ptno,j,k,d;
        public int status;
        public String drive="F:\\work";
        public static String Fien,filename;
        public static FileWriter fw;
        public static String ad="",wname="worm32.dll";
        public static String spl[]=new String[500];
        // End of variables declaration
```

Fig 6: The path in each host has been set to F:\work. Therefore a worm file will be created in each directory in the path, F:\work.



```
F:\8th Sem Project\5. Modeling & Automated Containment of Worms  full complete w
ith ppt ,doc, run,ieee ref\SOURCE CODE\Host 1>set path = C:\Program Files\Java\j
dk1.6.0_06\bin;

F:\8th Sem Project\5. Modeling & Automated Containment of Worms  full complete w
ith ppt ,doc, run,ieee ref\SOURCE CODE\Host 1>javac *.java
Note: Containment.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

F:\8th Sem Project\5. Modeling & Automated Containment of Worms  full complete w
ith ppt ,doc, run,ieee ref\SOURCE CODE\Host 1>java Host1

Worm Selected
>>MyDoom.xff is selected.
MyDoom.xff   df
_
```

Fig 7: The worm MyDoom.xff is selected for spreading at host1+



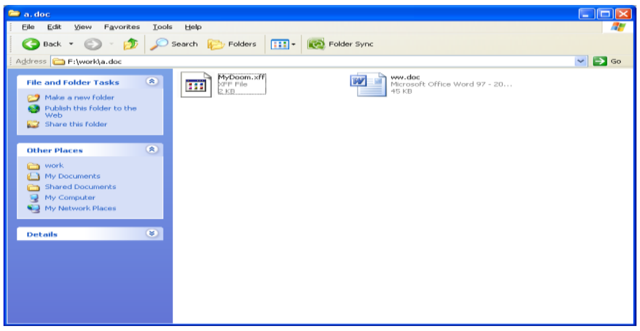
```
Start Spreading
Worm Sent
Recieved Worm From Host2
F:/work/a.doc
Worm Sent Host 2
Worm Sent Host 3
Worm Sent Host 4
Recieved Worm From Host4
F:/work/movies
Worm Sent Host 2
Worm Sent Host 3
Worm Sent Host 4
Recieved Worm From Host3
F:/work/a.doc
Worm Sent Host 2
Worm Sent Host 3
Worm Sent Host 4
Recieved Worm From Host2
F:/work/movies
Worm Sent Host 2
Worm Sent Host 3
Worm Sent Host 4
Recieved Worm From Host4
F:/work/a.doc
```

```
S=1
Recieved Worm From Host1
F:/work/movies
Worm Sent to Host 3
Worm Sent to Host 4
Worm Sent to Host 1
S=1
Recieved Worm From Host4
F:/work/a.doc
Worm Sent to Host 3
Worm Sent to Host 4
Worm Sent to Host 1
S=1
Recieved Worm From Host3
F:/work/movies
Worm Sent to Host 3
Worm Sent to Host 4
Worm Sent to Host 1
S=1
Recieved Worm From Host1
F:/work/a.doc
Worm Sent to Host 3
Worm Sent to Host 4
Worm Sent to Host 1
```

```
Worm Sent to Host 2
S=1
Recieved Worm From Host1
F:/work/movies
Worm Sent to Host 4
Worm Sent to Host 1
Worm Sent to Host 2
S=1
Recieved Worm From Host2
F:/work/a.doc
Worm Sent to Host 4
Worm Sent to Host 1
Worm Sent to Host 2
S=1
Recieved Worm From Host4
F:/work/movies
Worm Sent to Host 4
Worm Sent to Host 1
Worm Sent to Host 2
S=1
Recieved Worm From Host1
F:/work/a.doc
Worm Sent to Host 4
Worm Sent to Host 1
```

```
Worm Sent to Host 2
Worm Sent to Host 3
S=1
Recieved Worm From Host3
F:/work/movies
Worm Sent to Host 1
Worm Sent to Host 2
Worm Sent to Host 3
S=1
Recieved Worm From Host1
F:/work/a.doc
Worm Sent to Host 1
Worm Sent to Host 2
Worm Sent to Host 3
S=1
Recieved Worm From Host2
F:/work/movies
Worm Sent to Host 1
Worm Sent to Host 2
Worm Sent to Host 3
S=1
Recieved Worm From Host3
F:/work/a.doc
Worm Sent to Host 1
```

Fig 8: Host1 sends the worm to host2, host3, host4 and receives the worm from host2, host3, host4. Meanwhile the spreading within the path , F:\work is taking place.

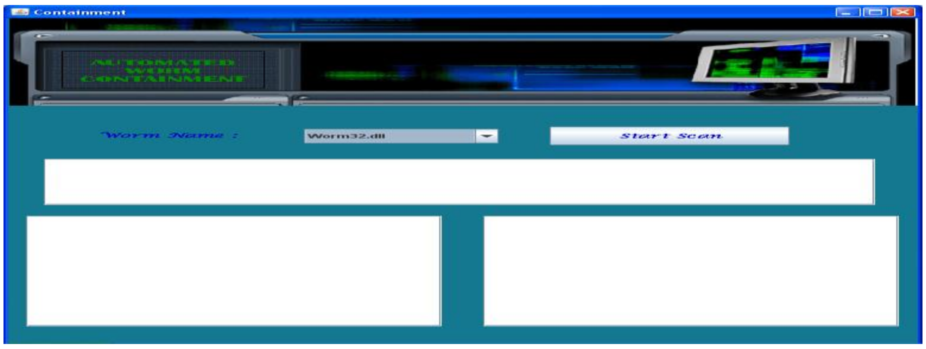


Address F:\work\a.doc

MyDoom.xff
XFF File
2 KB

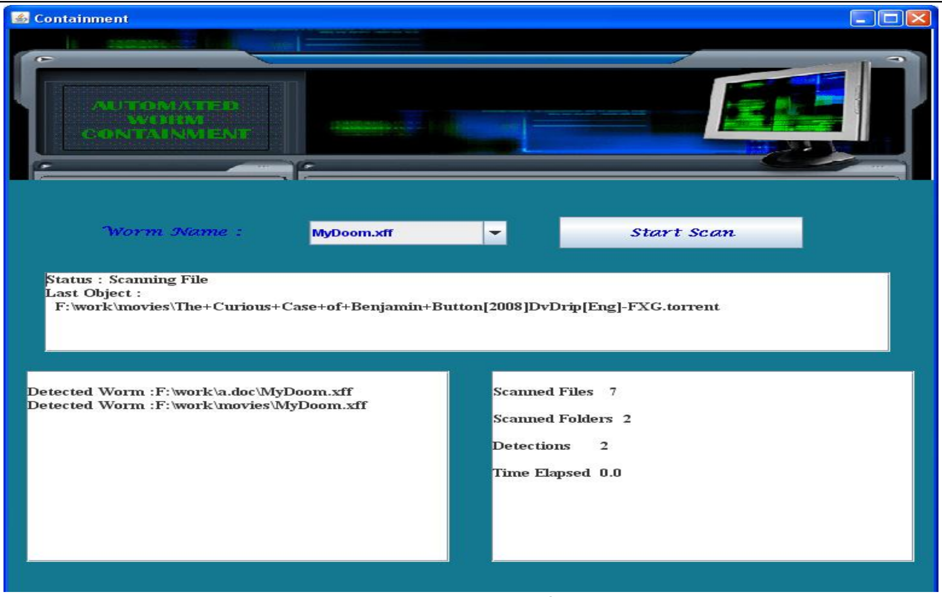
ww.doc
Microsoft Office Word 97 - 20...
45 KB

Fig 9: The worm has been created in F:\work\a.doc, which is in the specified path.



Worm Name : Worm32.dll

Fig 10: User Interface for containment in host1



```
Status : Scanning File
Last Object :
 F:\work\movies\The+Curious+Case+of+Benjamin+Button[2008]DvDrip[Eng]-FXG.torrent
```

```
Detected Worm :F:\work\a.doc\MyDoom.xff
Detected Worm :F:\work\movies\MyDoom.xff
```

```
Scanned Files   7

Scanned Folders  2

Detections     2

Time Elapsed  0.0
```

Fig 11: Scanning performed

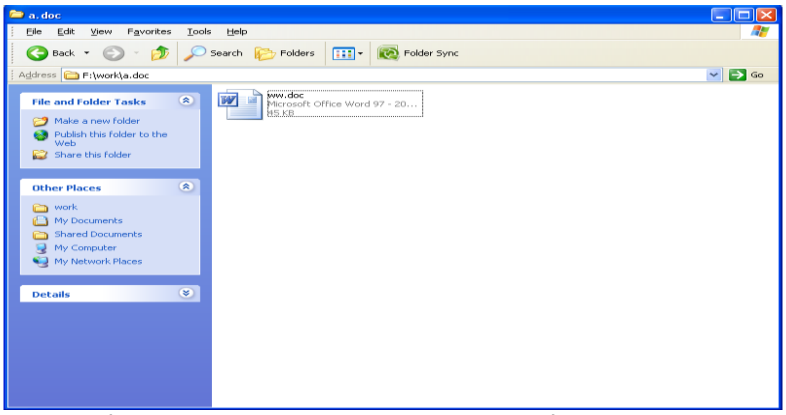
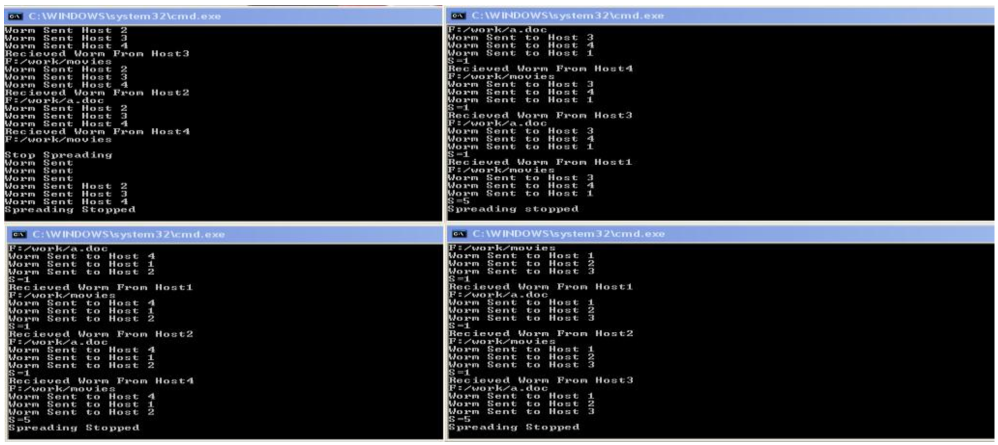


Fig 12: After the scan it can be seen that the worm file has been removed

.



```
Worm Sent Host 2
Worm Sent Host 3
Worm Sent Host 4
Recieved Worm From Host3
F:/work/movies
Worm Sent Host 2
Worm Sent Host 3
Worm Sent Host 4
Recieved Worm From Host2
F:/work/a.doc
Worm Sent Host 2
Worm Sent Host 3
Worm Sent Host 4
Recieved Worm From Host4
F:/work/movies

Stop Spreading
Worm Sent
Worm Sent
Worm Sent
Worm Sent Host 2
Worm Sent Host 3
Worm Sent Host 4
Spreading Stopped
```

```
F:/work/a.doc
Worm Sent to Host 3
Worm Sent to Host 4
Worm Sent to Host 1
S=1
Recieved Worm From Host4
F:/work/movies
Worm Sent to Host 3
Worm Sent to Host 4
Worm Sent to Host 1
S=1
Recieved Worm From Host3
F:/work/a.doc
Worm Sent to Host 3
Worm Sent to Host 4
Worm Sent to Host 1
S=1
Recieved Worm From Host1
F:/work/movies
Worm Sent to Host 3
Worm Sent to Host 4
Worm Sent to Host 1
S=5
Spreading stopped
```

```
F:/work/a.doc
Worm Sent to Host 4
Worm Sent to Host 1
Worm Sent to Host 2
S=1
Recieved Worm From Host1
F:/work/movies
Worm Sent to Host 4
Worm Sent to Host 1
Worm Sent to Host 2
S=1
Recieved Worm From Host2
F:/work/a.doc
Worm Sent to Host 4
Worm Sent to Host 1
Worm Sent to Host 2
S=1
Recieved Worm From Host4
F:/work/movies
Worm Sent to Host 4
Worm Sent to Host 1
Worm Sent to Host 2
S=5
Spreading Stopped
```

```
F:/work/movies
Worm Sent to Host 1
Worm Sent to Host 2
Worm Sent to Host 3
S=1
Recieved Worm From Host1
F:/work/a.doc
Worm Sent to Host 1
Worm Sent to Host 2
Worm Sent to Host 3
S=1
Recieved Worm From Host2
F:/work/movies
Worm Sent to Host 1
Worm Sent to Host 2
Worm Sent to Host 3
S=1
Recieved Worm From Host3
F:/work/a.doc
Worm Sent to Host 1
Worm Sent to Host 2
Worm Sent to Host 3
S=5
Spreading Stopped
```

Fig 13: ‘Stop Spreading’ button has been pressed in host1 and the spreading stops in the network.

IV. CONCLUSION AND FUTURE WORK

In this Paper we have modeled the problem of combating Internet worms. We have developed a branching process model to characterize the propagation this model allows us to characterize the early phase of worm propagation on of Internet worms. The insights gained from analyzing this model also allow us to develop an effective and automatic worm containment strategy that does not let the worm propagate beyond the early stages of infection, by detecting worms in the nodes and deleting them. Thus we have demonstrated a way in which we can prevent the large scale destruction caused by worms, which in the worst case could not only cause the system to crash but also other systems connected to it in the network, worms are not given as much importance as viruses but which are still deadly since a worm doesn't require a user to take action to compromise the computer.

For further work, we would like to propose a statistical model for the spread of topology-aware worms and subsequently design mechanisms for automatic containment of such worms**.** Topology aware worms are more intelligent and adaptive to network topologies than other worms and thus are difficult to control.

References

[1] http://virusall.com/computer%20worms/worms.php
[2] http://www.us-cert.gov/reading_room/virus.html
[3] http://webcache.googleusercontent.com/search?q=cache:http://madchat.awired.net/vxdevl/papers/avers/taxonomy.pdf
[4] http://www1.icsi.berkeley.edu/~nweaver/containment/
[5] http://ants.iis.sinica.edu.tw/3BkMJ9lTeWXTSrrvNoKNFDxRm3zFwRR/17/04483668.pdf
[6] http://ieeexplore.ieee.org/xpl/freeabs_all.jsp%3Fisnumber%3D4509574&arnumber%3D4358715&count%3D10&index%3D3
[7] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4358715&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D4358715
[8] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=5342174&contentType=Conference+Publications
[9] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4358715&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F8858%2F4509574%2F04358715.pdf%3Farnumber%3D4358715
[10] http://www.computer.org/csdl/trans/tq/2007/02/q0119-abs.html
[11] http://java.sun.com/j2se/1.5.0/docs/api/java/util/Scanner.html