CONTROLLED WORM REPLICATION
– ‘INTERNET-IN-A-BOX’

Ian Whalley
IBM TJ Watson Research Centre, PO Box 704, Yorktown Heights, NY 10598, USA
Tel +1 914 784 7368 • Fax +1 914 784 6054 • Email: inw@us.ibm.com

ABSTRACT

So-called ‘worms’ have been a feature of the malware landscape since the beginning, and yet have been largely ignored by anti-virus companies until comparatively recently. However, the near-complete connectivity of computers in today’s western world, coupled with the largely Win32-centric base of installed operating systems make the rise of worms inevitable.

The author will describe techniques and mechanisms for constructing and utilising an environment enabling the automatic analysis of multi-machine worms and network-aware viruses. Whilst these techniques are being developed for incorporation into the IBM/Symantec Immune System for Cyberspace, the paper is not intended to be a discussion of the Immune System concept. Instead, the intent is to describe an approach that has been applied to the problem with some measure of success.

The approach involves building a virtual SOHO network, which is in turn connected to a virtual Internet. Both the virtual LAN and WAN are populated with virtual machines. The suspected worm is introduced into this environment, and executed therein. The whole system is closely monitored as execution progresses in the isolated environment, and data is amassed describing what the suspected worm did as it executed. This data is then processed by the system in an attempt to automatically determine whether or not the suspect programming is performing actions indicative of a worm or Internet-aware malware.
1 INTRODUCTION

As is now well-documented,\cite{1,2,3} it is possible to create a functioning and practical system for automatically handling creation of detection and repair instructions for a significant percentage of viruses. This system (the ‘Immune System’) also handles distribution of the new detection/repair instructions to the clients in a scalable and manageable way. It is a subset of the process by which detection and repair instructions are produced with which this paper is concerned.

Whilst it is true that the Immune System performs well when it comes to producing detection and repair for the types of threat that it understands, it is also true that, from time to time, it needs to be ‘taught’ how to deal with new threat types. For example, at the time the Immune System was originally conceived, the clear and present danger in the area of viruses was from DOS file and boot infectors. Subsequently, the ability to handle macro viruses was added to the system – the system is designed in a modular fashion precisely to allow new components to be added without affecting the functionality of other components.

Several new components are at various stages of completion at the current time – this paper will touch briefly on the Win32 virus replicator, but will concentrate on the more interesting design and implementation challenges posed by a replication system for worms. This author has dubbed the system ‘Internet-in-a-Box’.

In addition, this paper will concern itself chiefly with the construction and deployment of a suitable environment for worm replication – nonetheless, it will discuss mechanisms used to determine what actions (if any) the program under test performed.

2 WHAT IS A WORM?

As with at least one of this author’s previous papers\cite{4}, the question of defining the type of malware under discussion is more vexed than one might have hoped. As with that previous work, there follows a brief list of previously suggested definitions of ‘worm’, in the context of malware:

[A worm is] a program that distributes multiple copies of itself within a system or across a distributed system.\cite{5,6}

Programs which are able to replicate themselves (usually across computer networks) as stand-alone programs (or sets of programs) and which do not depend on the existence of a host program are called computer worms.\cite{7}

A worm is an independent program which, when run on a computer, will attempt to infect other computer systems [...]. In this case the host program is the operating system of the computer, and the infected code is a stand-alone process or thread of execution running under the operating system.\cite{8}

The computer Worm is a program that is designed to copy itself from one computer to another, leveraging some network medium: email, TCP/IP, etc. The Worm is more interested in infecting as many machines as possible on the network, and less interested in spreading many copies of itself on a single computer (like a computer virus). The prototypical Worm infects (or causes code to run on) a target system only once; after the initial infection, the Worm attempts to spread to other machines on the network.\cite{9}
Worms are another form of software that, like viruses, spread by making copies of themselves. Unlike a typical virus, a typical worm spreads as a single self-sufficient program, rather than injecting itself into other programs. While there is general agreement that a self-sufficient program that spreads itself across a network of connected computers without the need for human intervention properly counts as a worm, there is less consensus on which of these features is the most essential to wormhood [sic]. In popular culture, novels such as Neuromancer by William Gibson refer to worms by the term ‘virus’, and the media has often used the words essentially interchangeably. There is currently no generally-accepted set of criteria for determining whether or not a given self-reproducing program is most properly called a ‘virus’ or a ‘worm’; both words, however, refer to programs that spread, and that therefore pose similar problems in the area of security.\[10\]

Readers who desire a possible formal definition of ‘worm’ are referred to endnote 11 – however, math-phobes (the author included) may rest assured that such mathematical treatments are not necessary to understand this intensely practical paper!

The marked definition is, in many ways, the most conceptually helpful. Whilst it is (inevitably) inexact in some ways, and tends to anthropomorphise the malware it is discussing, it also provides a clear and relatively concise definition. It does not explicitly mention the fact that most experts consider worms to be programs that do not infect other programs (in the generally parasitic manner of viruses), but this is strongly implied. However, other definitions make up for this omission. This author does not wish to spend time on suggesting another possible definition for ‘worm’ at this point, and will assume that readers understand the general concept.

3 BRIEF DESCRIPTIONS OF SOME WORMS

In order to understand the requirements of a worm replication system, it will be helpful to first examine (briefly!) the history of worms.

3.1 The Dark ages

We will begin by looking at three very early worms – whilst these do not map directly onto the current problem, they are helpful in forming a historical perspective.

3.1.1 Xerox

In the early 1980s, Shock and Hepps\[12\] conducted experiments with worm-like programs to perform computations on idle hosts on an early Ethernet LAN. Interestingly enough, the worm programs they were using would, occasionally, get out of control and have to be killed, which proved harder than expected, due to the worms’ ability to hop from host to host\[7\].

3.1.2 CHRISTMA EXEC

Another early worm was CHRISTMA EXEC, which was released in December 1987, and is well documented.\[5/7/9\] CHRISTMA EXEC required the user consciously to execute the infected email (which was written in REXX), and also required the most significantly afflicted company (IBM) to write what was probably the first ‘mail scanner’ in order to eradicate the worm from its systems.
3.1.3 Internet worm

In November 1988, the now famous RTM Internet worm was released – again, this worm has been well-documented in the past, and will not be covered in detail here.13/7

3.2 The Modern Era

The worms described above (see Section 4.1) are atypical of modern worms – this is almost inevitable, given their age! In recent years, new forms of worms have appeared – worms concerned with consumer-level protocols and applications – and it is these worms with which Internet-in-a-box is concerned. A brief description of some of these worms follows – this list is not intended to be exhaustive, it merely provides a glance at how the modern worm landscape has evolved.

3.2.1 IRC worms

In December 1987, the first well-known ‘IRC worms’ appeared. These worms relied on mIRC (in recent years, PIRCH has also become a popular target), and would spread themselves to the machines of people participating in the same IRC channels as an infected user.

Many of these worms permitted an attacker a primitive form of remote control over the infected PCs – the type of control later offered by so-called ‘Remote Access Trojans’ (RATs), but in a handy worm form.

3.2.2 Happy99

In January 1999, the well-known virus writer known as Spanska released perhaps his best-known creation – the worm known both as Happy99 and as Ska.14 (As an aside, the referenced article14 notes the difficulty in classifying malware such as Happy99/Ska: ‘It is not particularly easy to classify Ska. Whilst most virus researchers agree that it is a worm, others, including myself, have some doubts about this. The fact is that Ska cannot be classified as a real virus and it is not a traditional worm either. The general consensus is that its working mechanism shows more similarities with worms.’)

Happy99 worked by subverting the IP subsystem of Windows 95 such that it was able to recognise SMTP email being sent. It would then follow each message that the user sent with another message to the same recipient, containing a copy of itself. Even to this day, Happy99 is still widely reported.15/16 Happy99 works best in consumer situations – since it hooks SMTP mail transmissions, it will not work in many corporate scenarios – non-SMTP-based corporate email systems will not be affected.

3.2.3 Melissa

Melissa (which appeared at the end of March 1999) is another case of a piece of malware that is tricky to classify – even this author’s original article describing it called it a virus.17 The truth of the matter is that Melissa has two methods of spread – either as a traditional macro virus (by infecting documents as they are opened in the local Word installation), or as an Outlook-enabled worm.
It is, of course, the second method of spreading that made Melissa infamous, and enables it to be considered as a worm within the context of this paper. The Melissa-type of malware has since been referred to as a ‘virus/worm hybrid’.  

Melissa’s worm component, as readers will recall, worked by sending the currently active document to the first 50 people in each and every Outlook address book – however, this payload was only performed once. The upshot of this was that the first document that the user received was remailed, which in turn meant that it was usually the ‘seed document’ (an alleged list of passwords for porn sites) that was remailed.

Melissa quickly became the most widespread virus/worm in history – at least, up to that point in time. Estimates vary, but oft-quoted numbers claim that Melissa infected around 1,000,000 computers and caused ‘between $93 [million] and $385 million in actual damages’. Melissa was only widespread for a week – after that, incidents had dropped off very significantly.

### 3.2.4 ExplorerZip

ExplorerZip (or ExploreZip) appeared in July 1999 – again (as with Melissa) it relied on the user manually executing an attachment on an email, but this did not hinder its rapid spread. Unlike Melissa, ExplorerZip was ‘binary’ malware (that is to say, it was not written in a macro language – in fact, it was written in Delphi). ExplorerZip spread both by copying itself to unprotected SMB shares (Server Message Block – that is to say, Windows shares) on the LAN (and attempting to install itself into the Windows start-up procedure on any remote Windows installations it could access in this way), and by sending itself (in the form of a reply) to the originators of messages sitting in the user’s Outlook inbox. (The authors also discuss the difficulties of classifying the piece of malware they are describing: ‘Classification of ExploreZip is a grey area. A virus needs a carrier; an executable virus attaches itself to program files, a macro virus to documents. You could say ExploreZip attaches itself to emails, but this is not strictly true. The only email that it is ever attached to is of its own creation. Email is more the medium through which it spreads. Also, as an EXE file on an infected machine it is not associated with any other EXE file. However, you could say it infects WIN.INI files, but that is just to make sure it is always being run [sic]. On balance, we have opted for Worm rather than virus, but make up your own mind. Whatever you call it, it is definitely something you do not want on your system.’)

ExplorerZip was very widespread both on the Internet and within corporations – groups with unprotected network shares were particularly vulnerable to its payload, which involved destroying (by truncation) the contents of files with certain extensions.

### 3.2.5 Kak

February 2000 saw the arrival of VBS/Kak, which built upon techniques demonstrated by VBS/BubbleBoy – BubbleBoy was never widespread in the wild, but unfortunately the Kak worm was (and is).

Kak used the now infamous Scriptlet.Typelib vulnerability to cause itself to execute as soon as an Outlook user viewed an infected email. Kak still causes problems to some mail scanning systems, due to the fact that the infective code is appended to outgoing HTML email as the signature.
3.2.6 LoveLetter

In early May 2000, LoveLetter stole the coveted ‘most widespread malware in history’ crown from Melissa. LoveLetter\(^3\) (although written in VBS) is a Melissa-style worm – it relies on the user manually to execute an email attachment, whereupon it infects the local machine, and sends itself to every address in every Outlook address book. Worse yet, next time it executes, it checks for newly added addresses in the address books, and is able to send itself to those. LoveLetter is also IRC-aware, but this spreading mechanism is unimportant when compared to the mass-mailing functionality.

4 A LOOK AT THE PROPERTIES OF MODERN WORMS

Section 4.2 teaches us several things about the way modern worms behave, and the way that the definitions of ‘worm’ and ‘virus’ are not (at least in any useful way) mutually exclusive. These lessons are useful in designing a practical worm replication system.

4.1 Multiple techniques for spreading across LANs

Worms can spread across Local-Area Networks in a variety of ways – worms that have been seen so far (some of which are described above) have used the following techniques:

- Unprotected network shares
- Protected network shares (via password guessing)
- Already-connected network shares (via drive letters)
- Corporate email systems

4.2 Multiple techniques for spreading across WANs

The primary technique for spreading from organization to organization (across the Internet) at the present time is email – a well-designed worm replication system will be able to detect attempts to spread via other mechanisms, and raise an alert for the attention of a human analyst.

4.3 Other network considerations

In addition to the simple matter of spreading, malware has been seen which downloads components from remote Internet sites – the piece of the worm that initially arrives on the victim computer is a bootstrap system – in cases seen so far, this has had enough power to replicate. A good example is Babylonia.\(^2\)

5 REQUIREMENTS FOR A WORM REPLICATION SYSTEM

In particular, this section considers the basic services that must be provided by a worm replication system in order that it can work correctly and adequately.
5.1 Basic requirements

A functional worm replication system will have to appear to provide a number of Internet-style services from a wide variety of Internet hosts. In particular, the following services will be very relevant:

- HTTP (HyperText Transfer Protocol)
- FTP (File Transfer Protocol)
- IRC (Internet Relay Chat)
- DNS (Domain Name Service)
- Drive sharing
- Email
- Packet routing

6 PRACTICAL ISSUES

This section describes some of the more important areas in which Internet-in-a-Box differs from other Analysis Centre components – namely, single-virtual-machine virus replicators.

6.1 Network

To replicate worms, it is clear that a network is required. As suggested by the above definitions, a worm may well not exhibit any behaviour that can be considered ‘malicious’ with any degree of certainty in the absence of a network. Clearly, attempts to study worms (by execution) in an environment consisting of a single, non-networked, machine would be futile.

6.2 Interaction

Immune System components (such as Internet-in-a-Box) are expected to be able to run without any human intervention. The Immune System must run 24 hours a day, 7 days a week, 365 days a year, and consequently there can be no recourse to humans to reboot machines, shift network cables, reconfigure networking components, or install new operating systems. All such work must either be done ahead of time, or performed automatically as the system executes. Whilst this requirement is basically the same as for a single-virtual-machine virus replicator, it is much harder to achieve with a complex virtual network of virtual machines.

6.3 Client operating systems

Current Internet-in-a-Box development efforts are concentrated towards the various Win32 platforms, as this is where worms currently affect the computing population most significantly. In spite of this concentration of effort on Win32 platforms, the fundamental technology used in the Internet-in-a-Box component are by no means restricted to Win32 (as is shown later), non-Win32 operating systems provide critical services within the Internet-in-a-Box environment).
6.4 Provision of network services

As discussed in Section 5, a wide variety of network services (of both the LAN and WAN variety) must be provided in order to persuade a worm to replicate, or at the very least exhibit suspicious behaviour. These network services must be ‘faked out’ to some extent so that it is not necessary to provide access to the whole, real Internet!

7 ANATOMY OF A WORM REPLICATOR

7.1 Real or virtual?

There are two basic ways a worm replication system meeting the above requirements could be produced – either using real (physical) machines as the replication hosts, or using emulated (virtual) machines as the replication hosts. Both techniques have advantages and disadvantages – for the purposes of the IBM/Symantec Immune System, the virtual machine approach was chosen as being the most flexible, the best-suited to the already existing environment, and the most easily expandable. Research has been carried out on the physical-machine approach, but the primary research effort is now concentrated on the virtual machine approach.

7.2 Virtual machines, virtual networks, physical machines, oh my!

For the purposes of this paper, the machine on which the emulated sessions are running will be called the ‘physical’ machine; and the emulated sessions running on the physical machine will be called the ‘virtual’ machines.

A emulation package which supported flexible emulated networking was chosen – the package in question allows multiple virtual machines to run atop of a single physical machine, and have all the machines (both the virtual machines and the physical machine) accessible to one another via an emulated network (see Figure 1). The package also allows multiple virtual machines to be run atop multiple physical machines, and have all the machines (both the virtual machines and

Figure 1: A worm replicator with one physical machine
the physical machines) accessible to one another via an emulated network (which crosses a physical network in order to pass from physical machine to physical machine) – see Figure 2.

In Figure 1 and Figure 2, the key is the same – a solid box indicates a physical machine, a dashed box indicates a virtual machine, a dotted line indicates a virtual (emulated) network, and dotted/dashed line indicates a physical network.

It should be noted that, for the two physical machine implementation (Figure 2), the network traffic between the virtual machines must pass across a segment of physical network, in order to get from one machine to another.

In the practical implementation of this system, each physical machine should in fact have two physical network connections – one which is dedicated to carrying the traffic between the virtual machines (the potentially infected traffic), and the other to carry all other traffic – including control traffic, traffic between other machines, etc.

It should be clear to the reader that the system shown in Figure 2 can be expanded with more physical machines, as necessary to support the hardware load imposed by the required number of virtual machines – there is no need to stop at two physical machines.

In addition, it is possible for each virtual machine to have more than one virtual Ethernet card. This has not yet been required, although it could be useful at a later date for gaining more sophisticated control over packet routing.

7.3 What operating system runs on the virtual machines?

The OS of the physical machine(s) is not important for the purposes of this paper – it does not provide any undue support to the virtual machines other than being able to run the emulator and its associated services. However, the OSes of the virtual machines is clearly important, and worthy of some consideration here.

As discussed in Section 7.3, the primary operating systems running in the virtual machine environments are Windows – to be precise, at the current time, Win9x. The virtual machines are capable of running Windows NT and Windows 2000, and this will be brought on-line as time progresses.
The Windows virtual machines are all configured to log on to a Windows workgroup as the same (password-less) user. They log on automatically as the machine boots.

However, all the emulated (virtual) Internet services are also provided by a virtual machine – not the physical machine. The physical machine is kept isolated from events in the virtual machines and on the virtual network – no network connections are allowed from the virtual environment to the physical environment. The physical machine must be kept uninfected and unaffected by the potential malware executing inside one or more of the virtual machines.

Consequently, as mentioned above, emulated Internet services are provided by a Linux system running inside a virtual machine. In Internet-in-a-Box parlance, this Linux system is called the ‘Service Provider’ (see Section 9).

7.3.1 Virtual machine images

Once the virtual machines have been prepared with the relevant software and configuration, a ‘snapshot’ of each of their file systems is captured. This image serves as the starting point for future replication runs.

These file system images can be accessed directly by the operating system on the physical computer – in fact, they are manipulated by the ‘External Control’ (see Section 8.4.2) program in order to set-up for, and close down after, replication runs.

However, another operation needs to be carried out before the image is ready for use as a file system in a virtual client replication machine. All the files on the image are checksummed at this point – whilst the image is known to be in a clean state – and these checksums are used after each replication run to enable the External Control program to determine which files (if any) have been changed on a given virtual machine’s file system.

Figure 3: What software runs where?
7.4 Software running on the physical machine(s)

The reader should refer to Figure 3 to place the information in this section in context.

7.4.1 DHCP server

In order for the system to be dynamically expandable and configurable, it is necessary for the client machines (the Windows clients discussed in Section 8.3) to obtain their network configuration dynamically as they boot. Fortunately, DHCP (Dynamic Host Configuration Protocol) exists to serve precisely this purpose. One physical machine (and only one, even in the situation where there are multiple physical machines) runs a DHCP server, which provides virtual machines booting within the emulated environment with the following information:

- IP address
- Subnet mask
- Domain name
- Default router (gateway)
- DNS server

It is important to note that the last two items above both point to the Service Provider (see Sections 9.2 and 9.3).

7.4.2 External Control program

In addition, one of the physical machines (the same one as is running the DHCP server, for ease of comprehensibility) also runs program referred to as the External Control program.

This program manages aspects of the replication session’s execution – it sets up the images (see Section 10.2.1), it starts the virtual machines, it records information from the ‘Monitoring Subsystems’ (see Section 8.5.2) running within each virtual client machine, it tells the virtual client machines when to shut down (see Section 10.2.4), and it examines the images after the replication run is complete. As regards recording information from the Monitoring Subsystems running within the virtual client machines, the program performs this by connecting to the Monitoring Subsystems over the virtual network provided by the emulation systems (see Section 8.2).

7.5 Software running on the virtual client machines

In addition to the standard software on the clients (see Sections 8.3 and 9), the clients also run some custom monitoring and control software written in house to manage the attempted replication process, and to observe what happens as the suspect program runs. Once again, the reader should refer to Figure 3 to place the information in this section in context.

7.5.1 The Replication Controller

The ‘Replication Controller’ runs within the environment of the client virtual machines – its job is to execute the suspect sample (if the virtual machine in question is the ‘master client’ – see Section 10.2.1), to respond to pop-up windows, message boxes and the like from software run-
ning on the client, and to monitor the client to see if any obvious changes are taking place. In addition, the Replication Controller reports its current state back to the External Control program (see Section 8.4.2) via the Monitoring Subsystem – this ability is very useful in situations where the suspect program is inherently unstable (as often happens with possible malware).

The External Control program is able to allocate preset amounts of time for the Replication Controller to complete each given state – if the Replication Controller exceeds its time-in-state at any point, the system has failed and will be brought down. This system (of having time-in-state timeouts rather than a single overall timeout) ensures that, if and when the suspect program crashes the virtual client machine, the whole Internet-in-a-Box environment is brought down within a couple of minutes, rather than having to wait to make this determination.

7.5.2 The Monitoring Subsystem

Also running on the client virtual machines is the ‘Monitoring Subsystem’. This is a critical part of the whole replication environment, and takes the form of operating-system add-ons (for example, in Win9x, the Monitoring Subsystem is made up of several VxDs).

Currently, Monitoring Subsystem components have been written (on Win9x) to observe file system activity, and Registry activity. Network activity is to follow in a future version of the subsystem, as is Windows NT/2000 support. The Monitoring Subsystem effectively audits the virtual client machines.

As the Monitoring Subsystem works, it reports to the External Control program (see Section 8.4.2). The External Control program is responsible for recording the audit information provided by the Monitoring Subsystem – the Monitoring Subsystems allow the External Control program to connect (via TCP) to a port on the virtual client, through which the Monitoring Subsystem will transmit audit information. In addition, the Monitoring Subsystem will listen (via the same data connection) for instructions from the External Control program, which will either be processed locally within the Monitoring Subsystem (for example, the Monitoring Subsystem can be told to stop monitoring certain types of activity, to start monitoring certain types of activity, to shut down the client, etc), or be passed on to the Replication Controller for processing there.

For example, here is a sample trace from the file system auditing component of the Monitoring Subsystem:

```
Proc: Sample; FileOP: FindNext; Path: "C:\*.EXE"; SearchItem: "GOAT0015.EXE"; Return: YES
Proc: Sample; FileOP: Attr; FullName: "C:\GOAT0015.EXE"; Mode: SET_ATTRIB; Return: YES
Proc: Sample; FileOP: Open; Path: "C:\GOAT0015.EXE"; Mode: OPENEXISTING, READWRITE; Share: DENYREADWRITE; Return: YES
Proc: Sample; FileOP: Read; Path: "C:\GOAT0015.EXE"; Offset: 0; Length: 4096; Return: YES
Proc: Sample; FileOP: Seek; Path: "C:\GOAT0015.EXE"; Mode: From_Start; Position: 4096; Return: YES
Proc: Sample; FileOP: Seek; Path: "C:\GOAT0015.EXE"; Mode: End_Offset; Position: 0; Return: YES
Proc: Sample; FileOP: Seek; Path: "C:\GOAT0015.EXE"; Mode: From_Start; Position: 16914; Return: YES
Proc: Sample; FileOP: Read; Path: "C:\GOAT0015.EXE"; Offset: 0; Length: 0;
Proc: Sample; FileOP: Seek; Path: "C:\GOAT0015.EXE"; Mode: From_Start; Position: 16914; Return: YES
```
In fact, this trace is taken from a test run performed using one of the variants of the W95/Anxiety virus, and it shows the virus infecting a file called GOAT0015.EXE in the root of drive C: of the virtual client machine.

8 SOFTWARE RUNNING ON THE SERVICE PROVIDER

The Service Provider virtual machine is critical to the operation of Internet-in-a-Box. It can be regarded as the hub through which everything flows, and by which everything is monitored.

Currently, the Service Provider runs RedHat Linux 6.2, with a number of customizations. This section will outline some of those customizations, and how the clients are configured to take advantage of them. The reader should refer to Figure 3 to place the information here in context.

8.1 Samba

Samba is a free (under the GNU GPL), cross-platform SMB (Server Message Block) server – simply put, it allows Windows machines to access files held on Unix servers. It is highly configurable, extremely powerful, very flexible, and (most importantly for our purposes) easy to manage and monitor.

Samba on the Service Provider also acts as the ‘master browser’ – on a Windows network, one machine on each subnet is designated the ‘master browser’ (in fact, there are various types of browser – this level of detail is outside the scope of this paper, however; interested readers are referred to endnote 30) – this is the machine that will respond to requests from clients on the
LAN for lists of machines and their shares. For obvious reasons, when one considers worms such as ExplorerZip (see Section 4.2.4), it is necessary for browsing to be functional on the emulated network, or else the worm will not be able to find any shares to infect.

The Service Provider machine runs Samba (see Samba documentation), and deliberately runs it in a somewhat insecure fashion. It is configured to allow access without requiring password authentication (this is not the default behaviour!) to the user used by the virtual clients (see Section 8.3). This means that, as soon as the virtual clients have booted, they are able to access several shares on the Samba server without requiring manual intervention. The shares on the Samba server are preloaded with tempting looking files – files that viruses or worms might wish to infect, or (as with ExplorerZip) to delete. After the execution phase is complete, the shares will be examined to see if anything has changed.

8.2 DNS

DNS (Domain Name Service) is a critical part of the infrastructure of the Internet – it maps names (for example, whalley.org) to IP addresses (for the previous example, 208.13.42.8). For the purposes of Internet-in-a-Box, it is necessary to spoof DNS responses for all attempted name lookups that the worm may attempt.

Fortunately, this is fairly easy. The Service Provider hosts a copy of BIND (Berkeley Internet Name Domain – the most prevalent name-serving software in use today), and this software can be configured to return arbitrary responses to requests. In the case of Internet-in-a-Box, it is configured to return the IP address of the Service Provider machine itself in response to any DNS request it receives. And, as we saw in Section 8.4, the client machines are configured to use the Service Provider as a DNS server.

Consequently, the system can be configured to return arbitrary IP address in response to DNS queries from the virtual clients. In reality, there are a variety of different schemes that can and will be used to generate responses to DNS queries.

8.3 Routing

Whilst, in Section 8.2, we saw that the Service Provider is configured to redirect all name-based transactions to itself, this is not in itself sufficient to ensure that the Service Provider receives all traffic bound for outside the local-area network. It is possible (indeed likely) that worms will bypass the (possibly time-intensive) DNS lookup cycle by simply attempting to contact its destination directly by IP number. The reader will recall (from Section 7.4) that the Service Provider machine is configured as the default router for the client machines, and so will receive packets destined for such external hosts – the client believes that the packets will be passed out across a WAN in order to reach the intended destination. However, this is not what will happen.

Linux comes equipped with packet-level firewall and filtering utilities – primary amongst these is the ipchains utility. This allows Linux to provide firewalling and other packet filtering facilities (albeit fairly simple ones) to LANs. The Service Provider machine is configured to activate ipchains at boot-time, and ipchains is configured with some REDIRECT filters – such that all packets destined for outside the local subnet are redirected to the Service Provider machine itself.
Consequently, when the worm attempts to connect to port 12345 on IP address 208.13.42.8, it is in fact connected to port 12345 on the Service Provider machine (if there is a service running their to receive that connection, of course). The Service Provider machine is now pretending to be the whole Internet.

### 8.4 HTTP

Perhaps the most important user-level service offered on the Internet today is HTTP, otherwise known as ‘the Web’. Again, the Service Provider must provide HTTP service, and it must provide it as if it were the whole world, not just a virtual machine with as little software on it as possible. Consequently, the Service Provider machine runs Apache (the most prevalent Web server on the Internet today\textsuperscript{36/37}).

In this case, Apache is configured with a custom add-in module, which enables it to respond to any request in the affirmative and provide content. This module is currently fairly trivial, but before the system goes into production, it will be enhanced to ensure that the returned data is at least of the correct MIME type – currently, it simply returns a nearly-empty HTML page.

For example, here is a log of a transaction with the Apache running on the Service Provider from within the emulated LAN environment. Note that the client (in this case, telnet!) thinks it is talking to www.microsoft.com:

```
telnet www.microsoft.com
Trying 1.2.3.4…
   Escape character is ‘^[’.
GET /this/page/does/not/exist.html HTTP/1.0
HTTP/1.1 200 OK
Date: Thu, 13 July 2000 08:58:14 GMT
Server: Apache/1.3.12 (Unix) mod_perl/1.21
[… additional headers snipped …]
Content-Type: text/html
<html>
<head>
</head>
<body>
This is a dull page — not nearly as interesting
as <a href="http://www.whalley.org/">www.whalley.org</a>
</body>
</html>
```

(Readers should note that this is not the actual content returned by the Apache server on the Service Provider!)

### 8.5 Email

Another bastion of the Internet often used by worms is email – for example, see the discussion of Happy99 (Section 4.2.2). The discussion of email is (by necessity) more lengthy than that of previous protocols, and so is divided up into client issues and server issues.
8.5.1 Client-side email issues

In order to accommodate modern Outlook-enabled worms, the virtual client machines have a recent version of Outlook installed on them. On each client, Outlook is configured nearly identically – it is installed to interact with a POP3 (Post Office Protocol v3) server to receive inbound email and an SMTP (Simple Mail Transfer Protocol) server to deliver outbound email, as opposed to a Microsoft Exchange server. This is to obviate the need to have a second Service Provider machine running Windows NT or Windows 2000 (of course, when Microsoft provide a version of Exchange server which runs on Linux, this problem will go away). However, each client is configured with a different username (which will be used both to send mail to the server, and to receive mail from the server).

On all virtual clients, both the inbound (POP3) and the outbound (SMTP) mail server addresses are set to the Service Provider machine. Also, each copy of Outlook is configured with an address book containing the email address of other users within the Internet-in-a-Box world – this provides fodder for the large number of worms which send themselves to people in the accessible address books.

Finally, all the Outlook installations are configured to remember their passwords to access their mail on the server, in order to prevent untimely security-related pop-ups (needless to say, the author does not recommend that Outlook users configure their day-to-day Outlook installations in this fashion).

8.5.2 Server-side email issues

Clearly, it is necessary for the server email configuration to be compatible with the client email configuration – consequently, the Service Provider machine is equipped with both a POP3 (University of Washington IMAP/POP3/POP2 server) and an SMTP server (Postfix) to enable it to provide mail to the client machines, and to receive mail from them.

As was mentioned above (Section 9.5.1), each virtual client logs in as a different user – all these users have been pre-created on the Service Provider machine, and it is able to receive mail to and send mail from any of these users without additional configuration at runtime. In addition, each of the virtual users’ email inboxes comes pre-initialised with a couple of messages from other virtual users within the Internet-in-a-Box world – this is to provide fodder for worms like ExplorerZip (see Section 4.2.4) which respond to mail which the user has not yet filed away.

The SMTP package is configured to file all messages to anyone outside of the Internet-in-a-Box system in a special mail folder for later examination – that is to say, if a client attempts to send email to an unknown user, it will be preserved. Techniques described earlier (see Sections 9.2 and 9.3) ensure that, whatever the hostname of the mail that the client attempts to send, it will end up with an attempted delivery to the Service Provider machine.

8.6 IRC

Investigations with IRC services to run on the Service Provider machine are continuing – initially, it appears that some source code modifications will be required to enable an IRC server to run in the required controlled fashion. Currently, IRC functionality is not supported – however, the author believes that, were the situation to persist, it would be easy to add.
8.7 Network monitoring

In addition, the Service Provider machine is used to monitor the virtual network used by the Internet-in-a-Box system. This is necessary in order to be able to detect attempted network connections by programs running within the emulated environments to unsupported network services.

For example, if the suspect program attempts to make a connection to port 1701 on the machine with IP address 9.2.6.38, this connection will be redirected (by the routing component – see Section 8.3) to the Service Provider machine. However, the Service Provider machine is not listening on port 1701, and so the connection will fail. The easiest (and most flexible) way to determine that this connection was attempted (as well as to determine that other network-related incidents have occurred) is to run a packet monitor on the emulated network. At the current time, the packet monitor being used is the well-known and well-documented package TCPDump. If TCPDump logs are captured during the execution cycle of Internet-in-a-Box, and can (if necessary) be provided to the subsequent analysis phase.

9 BRINGING UP THE INTERNET-IN-A-BOX ENVIRONMENT

Now that the physical and virtual machines have been prepared, it is necessary to consider how to bring the Internet-in-a-Box environment up in a controlled and well-ordered fashion.

9.1 Physical machines

First, the physical machines are brought up – in fact, in a production environment, the physical machines will only ever be down for maintenance (for example, hardware upgrades, software upgrades, etc). These machines will also boot unattended, but they should boot very infrequently. These machines are controlled by a component of the Immune System referred to as ‘Dataflow’ – detailed information is available elsewhere, but for the purposes of this paper, suffice it to say that the Dataflow component provides work to the replicators and analysers as samples come into the system and as machines become available to process them.

9.2 Virtual machines

Once the physical machines have been integrated with the Dataflow system, Dataflow will allocate a suspect sample for a replication attempt. At this point, it is necessary for the External Control program (see Section 8.4.2) to determine what type of environment should be used to attempt to replicate the worm. This determination is an extremely difficult problem – in fact, it cannot be solved completely. At the present time, the system simply always uses the same environment – which consists of two Windows 95 machines (the virtual clients) in addition to the Service Provider.

9.2.1 Preparing the file systems of the virtual machines

As discussed above (see Section 8.3.1), the file systems of the virtual client machines are pre-
pared ahead of time to contain the relevant software and configuration information. However, it is of course necessary to inject the suspect program (the program which the system is to attempt to replicate) into one of the file systems of the virtual client machines.

Consequently, one of the virtual clients is chosen as the ‘master client’ – this is the virtual client which has the suspect file injected onto its file system. In addition, control scripts for the Replication Controller (see Section 8.5.1) are created and placed onto the file system of the master client. This injection process is performed prior to booting the virtual machines – the External Control program (see Section 8.4.2) is able to perform the injection operations using facilities made available by the operating system running on the physical machine.

The other virtual client machine is designated as the ‘slave client’ – this client will not directly run the suspect software. It can be regarded instead as a ‘goat machine’ – it exists solely as a tempting target for a worm to attack.

In order to render the goat machine more susceptible to attack, it is configured with file sharing enabled, and in an insecure fashion (fundamentally the same as the configuration of the Samba server described in Section 9.1). This makes the environment appear to be an insecure LAN, on which every machine’s hard drive is shared (with full permissions) to every other machine.

9.2.2 Bringing up the virtual machines

Once the file systems of the virtual client machines (both the master client and the slave client(s)) have been prepared by the External Control Program, it is necessary to bring up the virtual machines.

The Service Provider virtual machine must be booted first – this machine provides services that the virtual client machines require (see Section 8). The External Control program is aware of this requirement, and ensures that the Service Provider is invoked, and completes booting, before proceeding to boot the virtual clients. Once the Service Provider has booted successfully, the External Control program invokes the slave client(s). Again, these clients are allowed to successfully complete their boot processes before the overall process continues.

The final virtual machine to be booted is the master client (the client containing the suspect program – see Section 10.2.1). This client is started by the External Control program, which waits for it to boot and allows the Replication Controller program (running within the virtual client) to execute the suspect program.

9.2.3 Execution phase

At this point, the External Control program is connected to the virtual clients, and is receiving status reports (see Section 8.5.2) and audit information from the Replication Controllers as they execute within the virtual clients. This information is recorded for study by the subsequent analysis phase (or by a human analyst).

Inside the virtual master client, the Replication Controller has executed the suspect program, and is responding to pop-up windows and message boxes as they appear. In addition, the Replication Controller is driving applications hosted on the master client (most notably [at the current time] Outlook, but this work is ongoing), in an attempt to encourage the suspect program to exhibit worm- or virus-like behaviour.
As the process continues, the various Replication Controllers, associated programs running on the Service Provider machine, and the External Control program are all watching the process to see if replication or suspicious activity is occurring. Under ideal circumstances, the replication process is allowed to run to completion (in the absence of any bugs in the suspect program or suchlike) and the machines are brought down gracefully.

9.2.4 Bringing down the emulated machines

Whether or not the replication run completes successfully, the time eventually comes (hopefully fairly quickly!) to shut down the Internet-in-a-Box environment.

The shut-down process is, in essence, the reverse of the startup process. The master client is the first to be brought down, followed by the slave client(s), followed finally by the Service Provider machine. The shut-down process is initiated by the External Control program, which sends a shut-down code to the Monitoring Subsystem components on the virtual client machines and to a dedicated process running on the Service Provider.

Due to the fact that the system is infected (not to mention the fact that some of the virtual clients may in fact have crashed due to bugs in the suspect program or simply generic Win32 instabilities), the clean shut-down process described above may fail. When this happens to a real computer, the user turns the power off – this is, effectively, what the External Control program does. In fact, it kills the emulator processes that ‘contain’ the crashed Win32 sessions – it tries a variety of host operating system-based ways of doing this, ranging from nice to nasty. At the end of the shut-down process, all the emulated sessions will have been shut down or killed as appropriate.

9.2.5 Post execution analysis

Once the virtual machines have been shut down (whether gracefully or gracelessly), it is time to extract relevant data from their file systems. As described in Section 8.3.1, each virtual client’s file system was checksummed at the time it was constructed – each of the images used in this replication run is now re-checksummed, and the results are compared with the original checksums. Files that have been changed, or created, during the replication run are extracted from the images for analysis, along with information concerning which files (if any) were deleted. In the case of Win32 machines, Registry dumps are captured from the client file systems – these can also be used in analysis.

Treatment of the Service Provider virtual machine is slightly more complex – this machine’s file system is not simply compared (via checksum) with its state before the replication run commenced. Instead, the changes on this virtual machine are examined more intelligently. In particular, the TCPDump logs (see Section 9.7) are extracted, and the files on the available Samba shares (see Section 9.1) are compared with their expected states. Email Inboxes are examined (see Section 9.5.2). Other, miscellaneous, logs are extracted (for example, logs from the name server, the mail servers, the Web server, and the other various servers in place on the system).

The result of this post-execution extraction phase acts as both the input for the automatic analyser phase, and as notes for any subsequent human analyst. For example, in the case of ExplorerZip, the analysis phase will be presented with multiple copies of the same file (which
was created on each exposed share), and will determine that this particular piece of malware looks the same wherever it appears. The existing Win32 virus analysis tools are already able to make (and act upon) this determination. Other tools will examine the output from the extraction phase and note that this particular piece of suspect software has worm-like properties (in the case of ExplorerZip, files were created on machines which were not initially infected).

4 WORK FOR THE FUTURE

The reader should not be misled into believing that development work on Internet-in-a-Box is either at, or nearly at, a state of fait accompli – this is very far from the truth! This section outlines some of the future research directions to be undertaken with the prototype system, and some of things that need to be done before it can be integrated into the working Immune System.

4.1 Integrating Internet-in-a-Box with other replication systems

Perhaps the most obvious item of work that needs doing is to integrate the other (pre-existing) replication systems into the Internet-in-a-Box environment – in particular, the macro replicator. The Immune System already has a well-defined, well-understood and reliable replication system for macro viruses (for more information, see endnote 1 and the papers it references) – this replication system, however, exists outside the realm of Internet-in-a-Box. In order to handle macro worms (such as Melissa), it is necessary to execute the macro replication system within the Internet-in-a-Box environment. This work is already underway.

4.2 Analysis phase work

Many details of the post-execution analysis phase have yet to be precisely defined – the author will be the first to admit that there is considerable work still to be done in this area! Nonetheless, even if all the analysis phase is able to do (besides create a definition to enable detection of the worm) is alert a human analyst that something is definitely a piece of network-aware malware, this will be of some help. In an ideal world, the analysis phase would be able to write a description of the suspect program and the effects it had – whilst it is provably impossible to actually do this perfectly, it seems likely that it will be possible to provide some human-readable translation of the audit logs and associated data from the replication session.

4.3 Side-effect reversal

One area in which historical anti-virus products have been lacking is that of undoing the side-effects of the malware it is removing from your system. For example, the anti-virus product may be able to disinfect the virus completely from all documents, executables, boot sectors and whatever else, but all anti-virus products of which the author is aware leave the so-called ‘side-effects’ of the virus in place. For example, if the worm has created a value in the Win32 Registry to indicate that it has performed its mass-mailing phase, the current generation of anti-virus products will leave this value in place and consider the disinfection of the virus/worm to be complete nonetheless. With the audit tools available to Internet-in-a-Box, it will (in many cases)
be possible to automatically determine the side-effects of the suspect program, and to produce removal instructions for these side-effects.

4.4 Addition service monitoring and faking

It may be necessary to add emulation of (and logging of) more services than have currently been implemented as described in Section 9. Whilst the primary services have been implemented as a ‘proof-of-concept’, in the future both additional services, and additional functionality for existing services, will be required.

5 CONCLUSIONS

In this paper, the author has outlined a functional prototype of a worm replication system. This system, whilst not yet in production, will form an adjunct to the proven (and shipping) Immune System technology already produced, and will be able to provide valuable automatic worm analysis features for users of anti-virus products.

In the face of the near-inevitable increase in threats such as Melissa and LoveLetter, the author is confident that technology such as Internet-in-a-Box will prove to be as much a necessity in the future AV marketplace as automatic virus analysis and cure distribution technology is today.

6 ENDNOTES


16 Network Associates Virus Patrol reports, personal communications.
27 RedHat WWW site: http://www.redhat.com/
28 Samba WWW site: http://www.samba.org/
29 GNU GPL version 2: http://www.gnu.org/copyleft/gpl.html
31 Samba on-line documentation: http://us4.samba.org/samba/docs/
33 ISC BIND homepage: http://www.isc.org/products/BIND/
35 IPChains homepage: http://www.rustcorp.com/linux/ipchains
36 Apache homepage: http://www.apache.org/httpd.html
39 University of Washington IMAP Information Centre: http://www.washington.edu/imap/
40 The Postfix (formerly VMailer) home page: http://www.postfix.org/
41 The TCPDump public repository: http://www.tcpdump.org/