Distributed Computing with a Trojan Horse

Lauri Auronen, Antti Peltonen, Sami Vaarala, and Teemupekka Virtanen

Telecommunications Software and Multimedia Laboratory
Helsinki University of Technology, Finland,
lauri.auronen@hut.fi,
antti.peltonen@kolumbus.fi
sami.vaarala@iki.fi
tpv@tml.hut.fi

Abstract

Growing numbers of households are getting broadband Internet connections, forming an attractive platform for distributed computing. We describe how a trojan horse for covert distributed computing could be constructed: the mechanisms used, and how the user could be tricked to install the trojan using a known Microsoft Outlook Express vulnerability. The gains and risks of such a construction are considered. We found that such a trojan is straightforward to implement, even using a high level language such as Java. While installation of the trojan is platform dependent, the actual trojan can be run on any Java platform.

Keywords

Trojans, Trojan Horse, Distributed Computing

INTRODUCTION

Growing numbers of households are getting broadband Internet connections, forming an attractive platform for distributed computing (AeA 2003, Office of Telecommunications 2002, Nielsen//NetRatings 2003). On the other hand a large number of newly connected computers with homogeneous operating system installations have also attracted malicious parties. The amount of virus and worm activity is rising rapidly (ICSA 2002, Subramanya and Lakshminarasimhan 2001, Heinonen et al. 2003). The worms and viruses have thus far mainly concentrated on spreading mechanisms while carrying only limited payloads; some have had destructive effects, but mostly they have not done anything besides spreading (Moore et al. 2003, Kaspersky Lab 2003). However, a more destructive worm or virus might take advantage of the victim computer's computing power, creating a large distributed computing environment for the attacker.

In this paper we construct a fairly simple, but generic distributed computing platform that can be distributed as a trojan horse email attachment. We explain the mechanisms used to make the attachment look unsuspicious and how our distributed computing agent is installed on a user's machine. Lastly, we will discuss using the constructed distributed computing platform for finding the decryption key to a message encrypted with the DES encryption algorithm. The conceptual model of the computing platform is shown in figure 1.

We aim to demonstrate how easily such an agent can be constructed. We also try to point out the danger of having a large homogeneous operating system base, controlled by home users.

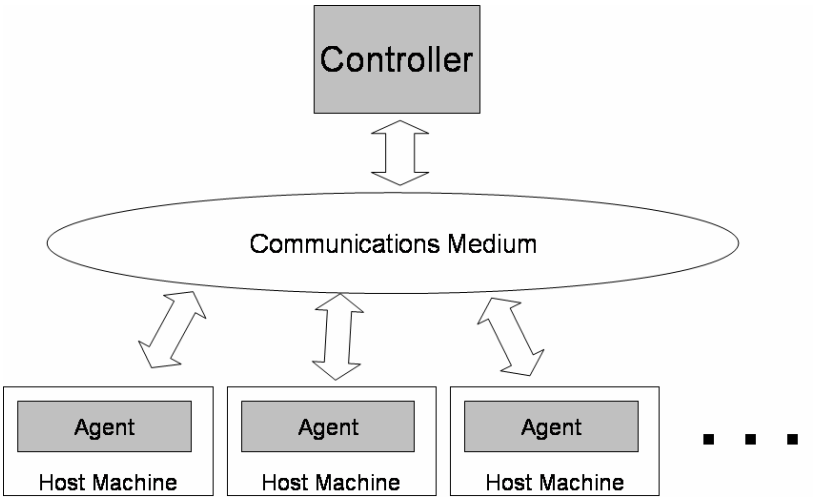

Figure 1. Conceptual model of a distributed computing platform. The communications medium could be, for example, an IRC network.

The rest of the paper is organized as follows. Section 2 describes the background and the needed technologies for constructing the agent. In section 3 we discuss the actual implementation. In section 4 we analyze the gains and risks of constructing a distributed computing network of such agents, and we also discuss how the trojan activity could be detected and blocked. Finally, in Section 5, we draw our conclusions of the subject and suggest directions for further research.

BACKGROUND AND CONCEPTS

In this section we briefly discuss the conceptual background and technologies needed for constructing, installing and running our trojan horse.

Email Vulnerabilities

Complex and feature rich email clients and operating systems are a mixed blessing. On the other hand they provide many useful things to users but at the same time they open up doors for hackers. We will take a look at two vulnerabilities: Scripts executed from email clients and the Three Extensions Vulnerability.

Scripting with VBScript

All Microsoft Windows operating systems include a component known as the Windows Script Host (WSH) used to create scripts of actions (Microsoft 2003). WSH scripts have the same access rights as the user executing them. Thus, if we are able to trick the user into executing a WSH script, we have access to all the user's resources. Such a script can also be sent as an email attachment and the receiver can choose to run the script directly from her email client.

Three Extensions Vulnerability

The three extensions vulnerability of Microsoft Outlook Express is a way of persuading the user to open a malicious file by making it look safe. The vulnerability can be exploited using a specially crafted suggested filename in the Content-Disposition field (MessageLabs 2003); a simplified example is the filename 'malware.jpg .vbs .jpg'. The first extension is visible to the receiver of the message, making the file appear safe to open. The second extension tells Outlook Express how to handle the attachment; '.vbs' means that the file is interpreted as Visual Basic Script. The third extension is used by Outlook Express to choose an icon for the attachment. The icon of a '.jpg' image makes the attachment seem safe to open. In addition to making the attachment to look safe, the vulnerability allows the mail to get past some content filters. (MessageLabs 2003)

Trojan Horses, Zombie Software, Computing Agents

Overview

A Trojan horse is malicious or harmful code contained inside apparently harmless program or data in such a way that it can get control and do its chosen form of damage (Pelaez and Bowles 1991). Trojan horses may be self-operating and execute some pre-destined malicious activity or they may open up a communications channel for their author to control them. A trojan can be installed on a single machine or a larger base of machines to obtain access to a larger base of computing power. A trojan can be included as a payload to a worm for larger scale distribution.

A large base of machines compromised with trojans has been used to launch distributed denial-of-service (DDOS) attacks (Yinjin 2001). Packet flooding is probably the most common use of the compromised machines, usually called zombies, but their computing power could also be used for any other form of computation (Geng and Whinston 2000).

Current and Possible Uses

Currently Trojan Horses are mainly used to install malicious software such as backdoors or zombie agents into victim computers. Backdoors such as Back Orifice and NetBus (Symantec 2003) are used to sniff user actions, retrieve passwords or steal files from victim computers. Zombie agents are software capable of sending real or spoofed packets to a selected victim. Zombie agents wait silently in the infiltrated computers for their controller to launch an attack.

Rather than performing aggressive attacks on hosts, Trojans can also be used more passively for e.g. covert computation. We demonstrate such use by harnessing our Trojan horse to break DES encryption keys.

Distributed Computing Using Other People's Resources

Utilizing Idle Computing Capacity

Most home computers are idle. Some efforts have been made to harness this great latent potential, such as search for extra-terrestrial intelligence in the Seti@Home project and breaking encryption algorithms (Seti@Home 2003, Gladychev 1998).

Distributed computing has been recently noticed also in the fields of biotechnology and finance. For example such companies as Parabon Computation and Entropia PC Grid Computing are offering distributed computing time for their clients. The price of such processor time is significantly lower than same processor time bought from a supercomputer. These companies also pay or are planning to pay for the leased processor time to the owner of the computer participating in their distributed computing projects.

The last group of programs that try to utilize idle computing time is the malicious software, doing anything from distributed computing to denial-of-service attacks. We discussed the current and possible uses of such software in the section Current and Possible Uses above.

Agent Distribution

In order for a distributed computing environment to be useful the number of individual agents performing computations should be as large as possible—we neglect controlling overhead for the time being. There are several mechanisms available for distributing the agents to clients.

One possibility is to directly install the agent software manually on each client. This requires access to the terminal and is therefore quite inefficient, and also poses severe difficulties with e.g. access control. A more efficient way of installing the agents is to do it through the network. The agent could be sent by email or put on a web site for people to download. In this way, a much larger network is achievable with the same time and costs.

One step further is to use an Internet worm for the distribution task. Such a worm could be used to obtain an agent base spanning up to tens of millions of hosts. The worm-like distribution method is discussed in detail in Staniford et al. (2001).

Agent Control Structure

Distributed computing is quite well researched. Previous work in the field includes such works as Nieuwpoort et al. on their Manta system (Nieuwpoort et al. 1999) and Gladychev et al. on Cracking RC5 with Java Applets (Gladychev et al. 1998). We will next give a brief introduction to these two works and compare them to our work.

Nieuwpoort et al. (1999) uses a modified version of the Remote Method Invocation (RMI) mechanism for communicating between their distributed computing nodes. However, their approach differs fundamentally from our approach. Their Manta system uses a collection of high performance parallel machines rather than workstation distributed geographically at random positions. The network they use for communication is different as well. While they use a fast dedicated wide-area ATM network, our network connection to the client can be almost anything from slow modem connection to fast T3 class connection. Also their system is somewhat hierarchical while our system is totally flat.

Much more similar to our work is the DISCO system by Gladychev et al (1998). The DISCO system uses Java applets for breaking a RC5 encryption key by distributed computing. Gladychev et al. pointed out four problems that are present when performing distributed computing on open networks. These are centralization, heterogeneous execution platform, dynamic runtime behaviour and lack of security. Both DISCO and our system use a centralized server - this restricts the scalability of the system. Heterogeneous execution platform raises the need for benchmarking the client. In the DISCO system this was taken into account, while we did not implement benchmarking for simplicity. Dynamic runtime behaviour is also an issue that DISCO and our system have in common. Dynamic runtime behaviour means that clients may leave and join the distributed computing task on the fly.

Large delay and variance in the network raise the need to reduce the control traffic between the nodes. This can be best achieved by making subtasks bigger, reducing control traffic and task setup overhead proportionately. Several methods for coordination of a large number of clients is discussed in Staniford et al. (2001).

Anonymizing the Controller

An important goal for an attacker is to avoid detection. Several techniques may be used. The attacker may use an anonymizing middle-man instead of direct client-to-controller connections. For instance, Internet news servers or peer-to-peer networks might be used. Control messages may be encrypted or obfuscated, or may even be hidden in ordinary messages using steganographic methods (Petitcolas el al. 1999).

We used Internet Relay Chat (IRC) (Kalt 2000a, 2000b) as an anonymizing middle-man in our experiment. IRC utilizes a basic client-server model where clients connect to servers to send and receive messages. The IRC servers are interconnected and exchange messages with each other. Messages can be sent directly to other users

or to IRC channels that users can join. IRC messages are textual and have a maximum length, so a controller needs to encapsulate and fragment control messages for IRC transport.

Some downsides of using IRC are: (1) IRC channels can be 'hijacked', (2) the default IRC server port is often blocked by company firewalls, and (3) IRC servers limit the amount of client traffic, which imposes some scalability constraints. Nevertheless IRC is useful as a model of a typical anonymization infrastructure.

Our scheme can be easily improved by the attacker. For instance, controller connections can be 'bounced' off clients: the controller connects to an existing client, which connects to the IRC server on behalf of the controller. This improves anonymization: even if the control connection was traced to the client, the client can easily hide any evidence of the bounced connection.

IMPLEMENTATION

Our distributed computing system consists of two parts: the server part and a large number of clients. The server divides the computation task at hand—ciphertext decryption, astronomical data analysis and so forth—into smaller units that are given to clients. We will first discuss the client side.

Agent Architecture

The agent consists of Java class files installed on the computer's hard drive. It includes a class called ClientContext which binds the client functionality together, and a control connection subsystem, an input handler subsystem, and a task runner subsystem. The control connection is a two-way text communications pipe. Any protocol can be used for the actual data transfer.

The input handler reads commands sent by the control connection and processes them. The control connection can also be used to relay data to and from the client, including Java class files. The input handler also includes dynamic class loading functionality for new commands and task classes. The mechanism is similar to the programmatic updates mentioned in Staniford et al. (2001).

The distributed computing part of the client software is built into the task runner subsystem. Each client includes one TaskRunner class and several task classes that are subclasses of the BaseTask class. When the TaskRunner is started it loads the last task and its state from disk and starts running the task. When the client software is stopped it again saves the current task and its state to disk. Through TaskRunner the controller of the software can change the task that is executed, give new data for the current task to process and set the priority at which the task thread executes. DESBreakerTask is our reference implementation of a client side task. It executes a known plaintext attack as a brute-force search for the correct decryption key.

Server Architecture

The server consists of a task manager, a control connection and a command line user interface with input handler.

The control connection sends data between the clients and the server, in our reference implementation using IRC. The command line user interface is used to issue commands to the server and it can also be used to directly command an individual client through the control connection.

The server can install new command and task classes to the client. For transfer over the control connection the class files are Base64 encoded and, in the case of the IRC connection, also divided into sufficiently small blocks that are sent in a delayed manner so that the IRC network will not drop them.

The distributed computing part of the server software consists of TaskManager and task classes. TaskManager is responsible for keeping track of the distributable tasks. All the actions that deal with the tasks are accessed using this front-end. TaskManager is also responsible for periodically firing up a checking process that checks that the distributable tasks do not have any subtasks that should already be finished and could be redistributed.

All distributable tasks should implement the interface DistributableTask. This interface includes the actions that are essential for all distributable tasks. KeyBreakerTask is our reference implementation for a distributable task, which searches through a key space for an encryption key. The third essential class in the task package is the SubTask. SubTask is a piece of the distributable task that is assigned to one client for processing. Subtask includes information such as the actual data to process, the client who is processing the subtask, and timestamps for task start, expected completion, and actual completion.

Installing the Client

Our implementation of the trojan horse is targeted for email distribution. In other words, each copy of the trojan is manually sent to a target machine. As discussed in section Agent Control Structure above, more efficient propagation methods are available. For distribution through email the compiled software is first packaged into a zip archive. In this way the several class files can be distributed more easily. The zip package is then

transformed into a self-extracting zip archive, voiding the need to have an unzip utility on the receiving computer.

WinZip Self-Extractor allows the creation of self-extracting archives that, when run, extract without prompting the user for any input and in the end execute a file extracted from the archive. This is very useful for trojan horse use.

Next, a specially tailored email utilizing the three extensions vulnerability is created. Attached to the email is a VBScript file (Figure 2) with two embedded files stored as Base64 encoded strings. The first file is an image that is used to mislead the user and the second one is the self-extracting zip archive. When executed, for example by double clicking the attachment in Outlook Express, the script Base64 decodes the image file and the archive file and writes them on disk. The image file is decoded and saved first and it is then shown using the system default viewer while the script continues to process the archive file in background. After the archive is extracted a batch file is run from the destination folder, which then starts the client software. Finally, the VBScript file adds a Windows registry entry which causes the client to start during reboot.

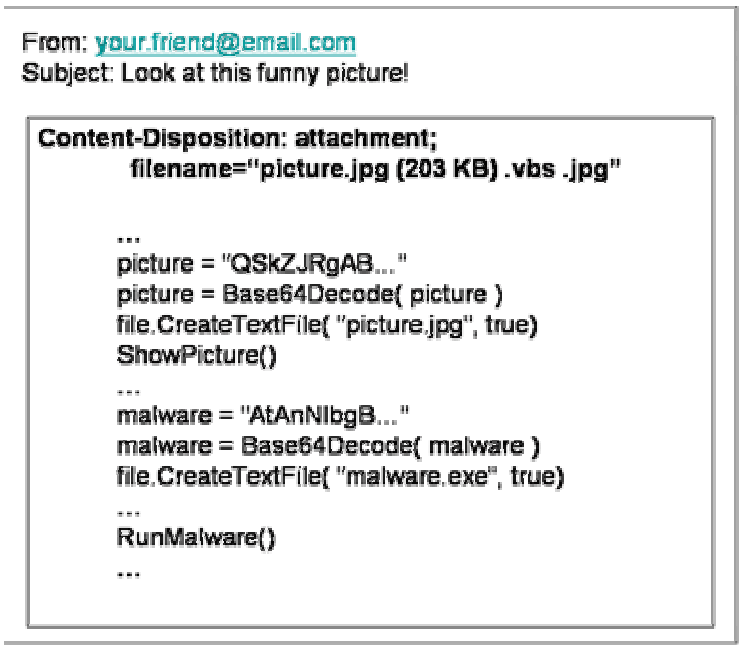

Figure 2. Outline of an email with malicious attachment

Utilizing the Client

If a given task unit is not finished in an estimated maximum time interval, the task unit is sent to a client again. A further improvement on the architecture would be to account for tampering of results. One possible protection is to send the same task unit to several agents (Gärtner 1999).

The reference implementation includes additional client-server communication to detect new clients in the network and to report tasks finished by clients.

ANALYSIS

In this section we analyze the gains and risks involved from the attacker's viewpoint. We also discuss how to detect and protect oneself against the attacks we used.

Attacker's Viewpoint

Gains

Our reference computer had AMD Duron 600 Mhz processor with 384 MB of memory running at 133 Mhz. Checking 20 million keys in one million key subtasks took 919 seconds from our reference computer. The average speed of processing one million keys is thus around 46 seconds. By using this result we can extrapolate the worst case time that is needed for breaking a 40 bit DES algorithm by brute-force known plaintext attack. By using mere 1000 computers concurrently to complete this attack would take around 14 hours.

There are several commercial networks that sell computing time or other processing on the network (for example, see http://www.gomez.com/?porivo=true). The controller could thus sell the processing power he gains from the covertly setup network.

Risks Involved

The risks from the attacker's viewpoint are divided into two categories: Legal risks resulting from possibly unauthorized use of computer systems (Lee 1990) and risks to the computation results from performing the distributed computing in an open environment.

Because the computation is performed covertly the probability of damage to the computation results is low. This is more of a problem in high profile distributed computing projects where status can be gained by faked results (Colley 2002).

The bigger issue might be the legal issues involved in utilization of computer systems without the owner's consent. Eventually the trojan horse software running on a computer will be noticed. Then the safeguard for the controller of the trojan is the IRC network. By analysis of the class files, the trojan horse can be tracked to the control channel in the IRC network. By listening to the traffic on the channel the controller can be detected. After that the controller's information can be extracted from the IRC server's logs. Thus there is a risk of being tracked down even with IRC used as an anonymizer, unless more levels of indirection are used.

Detecting the Attacks and Protecting Oneself against Them

We will next discuss how different elements of our Trojan horse can be detected and what actions one needs to take to gain protection against these attacks.

Covert Software Distribution through Email

If preview is enabled in the Outlook Express email client our script is shown as a broken image (red cross). This can give the user a hint that the attachment is not what it seems. The user may also choose to save the attachment to disk before opening it. In the save file dialog, the script file is always shown with the correct '.vbs' -extension.

If the user chooses to open the file it takes some time to Base64 decode the cover-up image file. Also, processor and disk activity continues after opening the image, which can be detected. When the batch file is run after extracting the WinZip SE archive, a command prompt window is very briefly visible on screen. However, more sophisticated malware would not be detectable this way.

Also, if the user is running good anti-virus software it will warn about malicious script activity. The best protections against this attack thus are using up-to-date anti-virus software with real-time protection capabilities and also to use an email client that does not have unpatched known vulnerabilities.

Three Extensions Vulnerability

This exploit can be detected by observing the name of the attachment file. In the attachment field after the file name there are many space characters and in the end of the line the string '....vbs(203KB)' or '...'.

Another way to detect this attack is when the file is opened. Depending on the level of patches installed, after opening the attachment you may get two confirmation windows that warn that the file may be suspicious. Answering 'no' to these questions will block the trojan horse installation.

The ultimate protection against this attack is to strip all email attachments. However, this is rarely plausible because of the legitimate uses for attachments. Another way to protect your computer against this particular attack is to simply not use Microsoft Outlook Express. However, other email clients have their own vulnerabilities.

Using IRC as a Signaling Channel

If the client, i.e. the trojan horse, cannot access its communications channel, the client cannot be controlled or used for distributed computing. Thus, a firewall that can block outgoing connections is sufficient to detect and protect against IRC communication. A good firewall configuration for an organization that does not utilize IRC for its operations should by default block all outgoing connections that are not explicitly allowed.

Another way to detect the control traffic is to monitor the system's network activity. Keeping an IRC connection live means sending data periodically to and from the server (Kalt 2000b).

The administrator of an IRC server used for such communication could also detect this attack by filtering traffic. Most of the data exchanges are very different from people chatting. For example, the statistical differences between subsequent client replies are small compared to natural language. The attacker could also make the detection more difficult by hiding the control traffic using steganography; the intent being to provide covert communication between two parties without exposing this activity to observers (Petitcolas et al. 1999).

Distributed Computing

Covert distributed computing activity can be easily detected by observing a computer's idle processor load. This can in Windows be done using the Task Manager. If there is a process that is taking a lot of computing time when the computer should be idle, there is reason for suspicion.

CONCLUDING REMARKS

We found in our experiments that creating a trojan horse application with a distributed computing component was fairly straightforward. We found that such a malware application can be created and executed covertly even in the high-level programming language Java. This relied on Microsoft's JView virtual machine being found on the Windows hosts used. It should be noted that only the install mechanism is platform dependent and the actual client application could be run on any Java enabled platform.

The distributed computing mechanisms implemented were simple; an attacker could implement a number of improvements to increase robustness and to make tracing more difficult.

Stealing processor time from computers is only one possible hazard caused by malware. A trojan horse payload could be more dangerous by distributing the user's personal data to the attacker, for example. Stealing processor time is a significant threat as it gives a monetary benefit to the attacker. We can conclude that an involuntary peer-to-peer file sharing network built by trojans is probable in the future. The upside of these threats is the raised security awareness of home users.

Further research could be useful, for example, in hierarchical control of the clients, incorporating self-propagation mechanisms to the client, and making installation of the software platform independent.

REFERENCES

AeA (2003), Broadband Subscriptions Up 488 Percent Since December 1999, URL http://www.aeanet.org/PressRoom/pret052203_Broadband2003Main.asp, Accessed 1 Oct 2003.

Colley, A. (2002) Cheats wreak havoc on seti@home: participants, URL http://www.zdnet.com.au/newstech/security/story/0,2000024985,20269509,00, Accessed 26 Mar 2003.

Petitcolas , F.A.P., Anderson, R.J. and Kuhn, M.G. (1999) Information Hiding – A Survey, *Proceedings of the IEEE, special issue on protection of multimedia content*, 87(7), 1062-1078.

Geng, X. and Whinston, A.B. (2000) Defeating distributed denial of service attacks, *IT Professional*, 2(4), 36-42.

Gladychev, P., Patel, A. and O'Mahony, D. (1998) Cracking rc5 with java applets; *Proceedings of the ACM Workshop on Java for High-Performance Network Computing (1998*).

Gärtner, F.C. (1999) Fundamentals of fault-tolerant distributed computing in asynchronous environments, *ACM Computing Surveys*, 31(1), 1-26, URL http://doi.acm.org/10.1145/311531.311532

Heinonen, A., Virtanen T. and Addams-Moring R. (2003), "We are running *what* ?!" – why the Slapper worm was able to spread in Finland, *Bill Hutchinson (ed)" Proceedings of 2nd European Conference on Information Warfare and Security"*, 0-9544577-0-6, MCIL, UK.

ICSA Labs Virus Prevalence Survey 2002 (2002) URL https://www.trusecure.com/download/dispatch/VPS2002.pdf?ECDE=W0107, Accessed 2 Oct 2003.

Kalt, C. (2000a) RFC 2811 - Internet relay chat: Channel management.

Kalt, C. (2000b) RFC 2812 - Internet relay chat: Client protocol.

Kaspersky Lab (2003) Virus Encyclopedia - Worm.SQL.Slammer (aka Helkern, aka Sapphire), URL http://www.viruslist.com/eng/viruslist.html?id=59159, Accessed 2 Oct 2003.

Lee, M.K.O. (1990), Hacking and computer viruses-the legal dimension, *Viruses and their Impact on Future Computing Systems, IEE Colloquium*, 6/1 -6/4.

Messagelabs (2003) Messagelabs - virus report - outlook express quirks being exploited by trojan writers 29 jan 2003, URL http://www.messagelabs.com/viruseye/report.asp?id=130, Accessed 1 Aug 2003.

Microsoft (2003) Windows script host basics, URL http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/wsconwshbasics.asp, Accessed 1 Aug 2003.

Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S. and Weaver, N. (2003) Inside the Slammer worm, *Security & Privacy Magazine, IEEE*, 1(4), 33 -39.

Nielsen//NetRatings (2003) Broadband Users Exceed 50% of All Household Internet Users, Nielsen//NetRatings NetView Audience Measurement Service Report on Japan Internet Ratings for April.

Van Nieuwpoort, R., Massen, J., Bal, H.E., Kielmann, T. and Veldema, R. (1999) Wide-area parallel computing in Java, *Proceedings of the ACM 1999 Java Grande Conference*, 8-14.

Office of Telecommunications (UK) (2002), UK reaches one million high speed broadband connections, URL http://www.oftel.gov.uk/press/releases/2002/pr58_02.htm, Accessed 1 Oct 2003.

Pelaez, C.E. and Bowles, J. (1991) Computer viruses, *Proceedings of Twenty-Third Southeastern Symposium on System Theory*, 513 -517.

Seti@Home (2003) SETI@home: Search for Extraterrestrial Intelligence at home, URL http://setiathome.ssl.berkeley.edu/, Accessed 5 Oct 2003.

Staniford, S., Paxson, V. and Weaver, N. (2002) How to 0wn the Internet in Your Spare Time, *Proceedings of the 11th USENIX Security Symposium*, 149-167, URL http://www.cs.berkeley.edu/~nweaver/cdc.web/

Subramanya, S.R. and Lakshminarasimhan, N. (2001), Computer Viruses , *Potentials, IEEE*, 20(4), 16-19.

Symantec Corporation (2003) Information on Back Orifice and NetBus, URL http://www.symantec.com/avcenter/warn/backorifice.html, Accessed 1 Aug 2003.

Yinjin@indiana.edu (2001) Ddos resources - basics of ddos, URL http://www.anml.iu.edu/ddos/howtowork.html, Accessed 1 Aug 2003.

COPYRIGHT