Global Information Assurance Certification Paper

Using Snort to Detect Rogue IRC Bot Programs

Christopher W. Hanna
Oct. 8, 2004
GSEC Practical Version 1.4c (Track 1 – Option 1)

Abstract

This paper provides an introductory guide for an IT professional to learn about and detect IRC bots on an internal network. The author is a systems administrator at a small Midwestern university who believes this research can be valuable to other educational or governmental organizations facing the same problems. This paper will first present a brief overview of what IRC is and how it functions, and why it is a potential threat vector. It will then show by example IDS rules via Snort that can be used to detect rogue IRC activity. Next, several packets taken from the University network will be analyzed to show how these rogue IRC systems communicate. Finally, some problems and solutions will be discussed with the hopes of improving the detection process.

What is Internet Relay Chat (IRC)?

In order to understand the problem, we first need grounding in the basics of what IRC is. We need to understand how IRC is designed and how compromised nodes are controlled before we can begin detecting them. This is only a brief look at some of the commands; a more in depth coverage can be found online, starting with RFC 1459, which defines the basic IRC commands.

IRC was initially designed in 1988, by Jarkko “WiZ” Oikarinen in Finland. IRC was designed as an expansion of existing BBS software of the time to allow real-time chat and USENET-style news (Stenberg). Jarkko expanded the chat portion and convinced users at other universities to join the experimental network. In a year's time, there were servers around the globe; the network expanded and grew into several forms, eventually ending up as the EFNet. Because of problems with duplicate usernames, the Undernet was formed as a competing network -- it allowed channel registrations and other amenities (Stenberg). The aforementioned RFC 1459 was written in 1993 as a basic description, and reported that the user base in the two years previous had seen a tenfold growth (Oikarinen/Reed). The 'NickServ' also existed at this time; NickServ was a service in Germany that allowed nicknames to be registered so users could keep their favorite nick when they logged back on. This service was shut down sometime in 1994. Due to a series of conflicts over bandwidth consumption issues, network 'splits', and registration problems, IRC has now splintered into hundreds of independent groups. There is no longer a monolithic IRC network from which all paths lead. Modern processing power and bandwidth availability, especially at universities, is such that almost any machine can host a minor IRC server, at least for around 1000 clients to use. In addition, the base IRC protocol from RFC 1459 is being constantly revised and added to, and as a result, a large number of people are developing custom additions to IRC (Stenberg).

IRC Terms

IRC is a client-server based system. Clients connect to a server and are identified by a unique 'nickname'. Though various operations, clients can then connect with other clients, and servers can connect with other servers, becoming clients of their own, like a relay. A special class of user called the 'operator' has the power to disconnect servers or users from the network.

This can cause what is called a 'split'; if two groups are on the same channel but one server is removed from another, the two groups can talk amongst themselves but cannot see anyone from the other group.

Clients organize themselves by joining channels. A channel is simply a sub-partition of the whole for which a client will receive messages, much like a television channel determines the programs viewed by its user. Channel names typically begin with '#', so the '#bicycles' channel is likely (but not necessarily) a channel for talking about bicycles. These channels are created by the first person to join them, and disappear if no one is in the channel. Each channel also has its own operators who can remove clients from that channel, change the title of the channel, or other tasks relating only to that channel.

Basic IRC Commands

The IRC commands covered here are ASCII strings, and this will simplify detection. Messages sent may or may not generate a reply; there is no guarantee of delivery in the standard.

The NICK and USER commands identify a user to the network. The NICK command sets the nickname of the user. This command may fail if that user already exists on the network, and the client will be forced to choose another name. The USER command identifies the host, server, and screen-printable name of the user. Since clients must issue these commands in the beginning phases of an IRC connection, looking for this string on the network will identify IRC bots trying to log into a server or a server being logged into, depending upon the direction of the traffic.

Several messages deal with channel operations. The JOIN command allows a user to join a specific channel. The MODE command allows modifications to made to a channel; for example, for it to be made private. The MODE message can also be used on a specific nickname to change its status.

PING and PONG are two commands used to maintain connection between the server and the client. If no activity occurs on a connection, the server will send a PING to the client to make sure it is alive. The client must respond with a PONG within a set time interval to remain active.

Users can contact other users with the PRIVMSG message that allows one user to send a private message to another. Within the scope of this paper, PRIVMSG is usually used to send the controlling entity an indication that the compromised client machine is ready for their use or has completed some task. Often these nodes are called ‘zombie’ machines, because they are mindless drones waiting for marching orders from the central authority. A large group on these clients can be controlled by one server: authorities in Singapore found a group of 10,000 zombie machines connecting to one central server, waiting to be used as part of an attack (Roberts). We will be examining the details of this traffic later.

Clients can also open up a direct connection to each other, bypassing the IRC server entirely once the connection is established. This is normally done to speed up data transfers. IRC provides the DCC message for this operation. These messages rely on CTCP, or Client-To-Client Protocol, which is not covered here (see http://www.irchelp.org/irchelp/rfc/ctcpspec.html for a good description). A DCC message from one user to another includes the type of connection (CHAT or SEND) as well as the address and expected

port that the host connection is expected to listen on. In analysis of the IRC bot packets, it appears that the DCC SEND command is commonly used to transfer files between hosts. The term 'XDCC' is also thrown about – this is not a message type but instead a mechanism by which a script can initiate a DCC transfer. Several channel monikers on IRC use XDCC as part of label, making it a useful string to search for when we generate rules later on.

Finally, the CONNECT message is used to order one server to log into another. This can allow a smaller IRC server to act as a gateway to a larger one. There are many more IRC messages. The standard reference for these commands is RFC 1459, but some IRC servers have additional commands that have been added since this standard was written. We should also note that the messages above are what is seen at the network level and not the actual commands used in an IRC client; instead commands in a client will be prefixed with '/', such as '/privmsg Mordecai Is that you?', which would send 'Is that you?' to the user registered with the 'Mordecai' nickname.

What is an IRC 'bot'?

Put simply, a 'bot' is an automated program that sits in the channel like a user and acts on messages sent by others from the channel. As the state of permissions and channels in IRC is transitory in nature, some use a bot program to keep control of a channel while they are away. For example, an IRC bot could give a user operator permission on a channel as soon as they log in, provided they PRIVMSG it a certain password. This allows the channel to stay in their control at all times. One example of a flexible bot program is the Dancer bot, which helps keep an IRC channel safe from flooding and performs other services such as spell checking, dictionary lookups, and SMTP VRFY checks (Holst). IRC bots can also act as a simple responder, dispensing fortunes, channel logs, URL links, or even movie times to response to PRIVMSG commands from users or requests made from those chatting in the same channel as the bot.

However, in the cases we examine here, the purpose is not so benign. Commonly these bots will also serve as an automated file transfer service, or to collect information from compromised machines, which in turn have a trojan horse program installed that forces them to quietly join the channel. The joiners are themselves bots, being automated programs that are listening for a command to come across the channel so they can react to it. For example, a hacker could order a channel of 500+ bots to commence a distributed denial-of-service attack by giving a simple command on the channel to which they would all be listening to. The 500 hosts involved could be from 500 different environments, meaning the denial-of-service attack would have a higher likelihood of being successful. Because of the problems associated with bots and potential for flooding or misuse, a lot of the primary IRC servers, especially in the United States, aggressively ban users who run them. EFNet in particular does this, and one Undernet administrator adds any bots he sees to a kill list, and believes that bots are “generally a nuisance”. (Wagner). There is, however, no reason why an intruder cannot simply connect to another IRC server or even keep their own IRC server off of the rest of the IRC network. As we will see later, an attacker can set up an IRC server on one of the compromised machines if they judge it is adequate to hold the traffic. The distributed nature of IRC serves well to meet this need.

Why should my organization be concerned?

Rogue IRC bots can pose a multitude of risks. Let us look at the simplest case and assume an organization is housing only one of these infected machines. This machine is, by nature, compromised, carrying with it all the usual risks of data compromise, espionage, and loss of time and productivity cleaning up the problem. However, machines compromised with IRC bots are usually designed to act as part of a large network with one or more controlling points. As part of that design, attackers usually use the bots themselves to act as scanning agents to find more machines with weak security in order to set up more bots, and so on. Consider any computers in your enterprise with weak security likely targets, and any commonalities between machines in the local environment will likely be exploited almost immediately. Intruders have even been known to fight amongst themselves to divide up their share of the territory -- “Do Not Rehack”, displays a typical Serv-U FTP server banner associated with IRC compromise, as if there were some ethical bounds to exploitation. Using this exploitation software is not even that difficult, and hackers target university and government sites, because they typically have lax security and fast network connectivity (Graham). Even as far back as 1998, the Computer Incident Advisory Capability of the Department of Energy warned about this growing issue (Rayome).

A larger problem presents itself when the enterprise gets scanned from within for vulnerabilities and an attacker acts on those vulnerabilities. Much of this can be automated with scripts, which are relatively simple programs. An attacker does not need to even be able to write or understand this process; they may just download an IRC bot kit off of a website or through other contacts. As this network of compromised machines starts to grow, so does the personnel time that must be committed to cleaning up the problem.

Once a few machines on the internal network become compromised and are set to scan, consider the majority of the enterprise scanned if some network segmentation is not in place. This will be a very quick process, and includes port-scanning, exploitation of services, and cracking of passwords. As part of the scanning, there may be loss of network connectivity as routers or firewalls struggle to keep up with the large quantity of requests. As seen later, some attackers are now scanning more slowly in an attempt to evade detection. Also, most common workstations are now powerful enough to overwhelm a basic router when acting as a group. As more nodes are added to the system, more scanning takes place. Larger scale attacks can now occur; and a coordinated attack of fifty nodes from your network at another organization's network is not going to be taken lightly. The liability can grow exponentially if this issue is not dealt with as soon as it is detected.

Problems with detecting IRC bots

Unfortunately, many of these IRC bots pass by undetected until they become a significant problem. There are several reasons for this. First, they do not follow the same pattern of infection or signature from incident to incident. IT organizations which are used to dealing with known quantities such as a virus or security hole may be surprised by the flexibility of this problem. Support staff may be aware of the symptoms but not the causes. Secondly, some stateful firewalls, both hardware and application, might not alert about this traffic since it is initiated at the client side once compromise

has taken place. Lastly, IRC bots are unusual in quantity of packets sent. The actual amount of data sent can be very minimal until the bot is put to use in a denial-of-service attack or other ends. If the organization relies on a list of top bandwidth consumers to identify security problems, IRC bots may not be displayed on that chart until the worst case scenario hits, potentially knocking out the entire network.

Brief overview of Snort

Snort is an open source intrusion detection system. An intrusion detection system, or IDS, aims to monitor network traffic and look for signs that intrusion is taking place. It then dumps these 'alerts' into some form that security personnel can use to follow-up on the traffic. Snort, like most IDS systems, is configured with a set of rules or signatures to log traffic which is deemed suspicious. There are several factors when setting up a Snort system such as horsepower, listening capability, maintenance, and positioning within the network. The actual Snort installation is fairly simple and several guides are available:

"Snort, Apache, PHP, MySQL, ACID on Redhat 9.0 Installation Guide"
http://www.snort.org/docs/snort_acid_rh9.pdf

"Snort Install on Win2000/XP with Acid, and MySQL"
http://www.sans.org/rr/papers/index.php?id=362

"Snort Alert Collection and Analysis Suite"
http://www.sans.org/rr/papers/index.php?id=1253

Snort rules are used in this paper because the Snort suite consists of free tools that anyone can use on multiple platforms, so the system has become a standard. Rules passed along via mailing list or other avenues are often coded for Snort.

Sample rules for detection

Snort rule generation is both simple and difficult. The key to a good Snort rule is properly defining the signature you wish to match. Rules that catch too many packets will lead to false positives -- and overwhelm staff who read the alerts, by deluging them with possible problems which do not exist. On the other hand, rules that are defined too tightly will miss some types of exploitation completely. Additionally, attack vectors change from day to day, so the rules database needs to be monitored, updated, and 'weeded' frequently. As a beginning step, let us look at a generic rule which has nothing to do with IRC bots, at least not directly.

alert udp any any -> any 69 (msg:"TFTP GET nc.exe"; content: "|0001|"; offset:0; depth:2; content:"nc.exe"; offset:2; nocase; classtype:successful-admin; sid:1441; rev:2;)

The first part of this rule, "alert udp any any -> any 69", defines the basic parameters that we match. In this case, we are only looking at UDP traffic from any host to any other host with destination port 69. This happens to be the port and protocol associated with TFTP transfers. TFTP actually is a

common element used by IRC bots, as infected nodes transfer files to each other. However, TFTP is also used by many legitimate devices. Keep in mind that the IDS has to be able to listen between these nodes in order for this rule to work; having the IDS at the network perimeter is not going to help detect traffic between two internal workstations. All of the packet logs in this paper are from the network perimeter, but we will see traces of activity within the enterprise that may have gone undetected if we were not looking for IRC bot activity.

Because we do not want to match just any TFTP packet, the next part of this rule defines some extra parameters to our alert. The 'msg' is the text message displayed in the database, so the personnel reading the IDS alerts understand what the alert shows. This description should be brief and meaningful. The 'content' field contains the packet signature to track, or the actual content of the packet that we are attempting to match. The 'offset' indicates the byte offset where the pattern matcher should start searching for the pattern. The counterpart to this directive is ‘depth’, which specifies the last byte in the packet that needs to be searched for this pattern. In the above, there are two separate patterns that must be matched for the alert to trigger. This rule is looking for a TFTP GET (content: “|0001|”) in the first two bytes, followed by the text ‘nc.exe’ somewhere after. The nocase directive specifies case is not important to the match. Finally, the classtype indicates what class of alert it is filed under, whereas the 'sid' and 'rev' give each rule a unique ID and revision number.

We should also note the “->” arrow, which indicates the direction of traffic. We may want to match traffic in either direction (“<>”) or only to or from our internal network. In the snort configuration file, one can define the internal network like so:

var HOME_NET 139.102.0.0/16

Then, rules can include this parameter – it is much simpler than putting our network segment in every rule. Incidentally, 'nc.exe' is a program called Netcat which was the staple of Windows compromises for years, which is why this alert exists in the first place.

Having covered a sample rule, let us look at some actual rules to detect IRC bots on the network:

alert tcp $HOME_NET any -> any 6666:7000 (msg:"Possible IRC access (JOIN)"; flow:to_server,established; content:"JOIN"; classtype:misc-attack; sid:1000041; rev:7; tag:session,30,seconds;)

This rule tracks connections from the internal network to the external network on all TCP ports in the range 6666 to 7000, as long as the packet contains the word “JOIN”. IRC typically uses port TCP 6667 for communication, but that is not a guarantee. Many IRC servers use a slightly modified port number, such as TCP 7000. In fact, there is a problem with this rule as it relates to intrusion. Most exploiters do not code their exploits themselves; attackers instead find a pre-packaged “kit” to download and then customize it to suit their needs. The IRC port number is just a configuration option in that kit. Many intruders are now using varying ports as a means to escape detection where the others have been caught. To remedy this problem, suppose we

open up this rule to catch IRC servers on any port:

alert tcp $HOME_NET any -> any any (msg:"Any port possible IRC access (JOIN)"; flow:to_server,established; content:"JOIN"; classtype:misc-attack; sid:1000042; rev:7; tag:session,30,seconds;)

It may not be clear at first, but this is not a good rule – if the alerts that this rule generates are viewed, they are almost entirely false positives. Why? Since this rule matches any packet with "JOIN", it matches any web pages with the word, unencrypted e-mails with the word, and so forth. As mentioned before, writing rules is a delicate process. This rule has to examine every outgoing packet, no matter what the source or destination, so it is also very processor consumptive for the sniffing machine. As the alerts themselves need to be analyzed to see if they are useful and proper, the sniffers themselves needs to be analyzed for CPU and memory consumption to ensure that the rules are efficiently written. If a sniffer does not have enough process time to run every rule per packet, it will begin to miss packets. This is one reason why multiple sniffers can be very useful when working in anything but a small environment.

How can this rule be made more specific? First, a pattern can be matched that does not also match a common English word. The pattern can also match known bad traffic that has been observed. If we can sniff the traffic of a known compromised host, we can use that data to formulate possible avenues of detection. This is a good idea not just for this rule, but for most areas that the IDS will cover. In the case of IRC bots, there actually seems to be a good deal of conformity in the kits used, as just some specifics have been tweaked. Unfortunately the kits can change fairly quickly. Here are two rules that are the result of analyzing known systems:

alert tcp $HOME_NET !21:443 -> any 1000:65535 (content:"PRIVMSG"; nocase:; content:"Exploit"; nocase:; within:80; tag:session, 20, packets; msg:"Possible RogueIRC 03"; classtype:trojan-activity; sid:1000168; rev:6;)

alert tcp $HOME_NET !21:443 -> any 1000:65535 (content:"PRIVMSG"; nocase:; content:"lsass"; nocase:; within:80; tag:session, 20, packets; msg:"Possible RogueIRC 04"; classtype:trojan-activity; sid:1000168; rev:6;)

These rules were distributed via the Educause security mailing list (Holstein). They focus on traffic sent from any port locally except 21 thru 443, outward to high numbered ports. This does mean that a crafty attacker using port 80 to send packets will not be detected; in this case, we are willing to take that risk because the overhead in looking through port 80 traffic is so high. We might also have another rule that filters this port more effectively, or use a web proxy to help filter traffic instead. Each logged packet, in order to match one of these two rules, must contain a "PRIVMSG" followed closely (within 80 bytes) by either "Exploit" or "lsass". These rules were developed because activity was seen matching these rules. The "lsass" rule refers to a Microsoft security vulnerability regarding the "lsass.exe" program (for more details, see http://www.microsoft.com/technet/security/bulletin/ms04-011.mspx). It is clear these are temporal rules that may not be effective for

very long. When major new patterns of attack emerge, new rules to deal with the new exploits must be developed quickly.

Instead of broadening our scope from the standard IRC ports and working outward, we can look at the inverse of normal traffic; that is, traffic of this type on atypical ports. The following rule was formed from the 'bleeding edge' rule set (Esler).

alert tcp $HOME_NET any -> $EXTERNAL_NET !6661:6668 (msg:"BLEEDING-EDGE IRC - Nick change on non-std port"; content:"NICK "; offset:0; depth:5; nocase; dsize:<64; flow:to_server,established; tag:session,300,seconds; classtype:trojan-activity; sid:2000345; rev:3;)

This rule proves quite fruitful as long as we do not detect many false positives from this range. It also tracks only in one direction, which saves some processing time if we feel that the internal network is mostly safe or that another rule is catching internal IRC servers.

We have looked primarily at cracking and exploitation, but we should also consider basic piracy, whether part of a compromised system or not. IRC bots can be used to deliver this traffic as a kind of peer-to-peer network, with most of the actual peers being compromised machines. In some cases, a person will set up a bot on their own machine for the purpose of sharing. Usually the best way to focus on this problem is to look for XDCC packets. Here are two sample rules that were written based on university traffic, one for any XDCC activity with the word 'movie', and another for 'rar' files which are typically compressed software.

alert tcp any 6666:7000 -> any any (msg:"Possible XDCC Activity [MOVIE]"; flow:to_server,established; content:"XDCC"; nocase:; within:80; tag:session, 20, packets; content:"movie"; nocase:; classtype:misc-attack; sid:1000442; rev:7;)

alert tcp any 6666:7000 -> any any (msg:"Possible XDCC Activity [rar]"; flow:to_server,established; content:"XDCC"; nocase:; within:80; tag:session, 20, packets; content:".rar"; nocase:; classtype:misc-attack; sid:1000142; rev:7;)

The goal here is not only to find the problematic machines, but to gather some rudimentary evidence that can be used to show what is being delivered. There is some legitimate IRC traffic in the universe, and if we are going to allow the protocol we need to weed out the good (few) from the bad (many). If the filename is in the XDCC request, we should be able to find the destination server, source server, as well as the file being transmitted from the alert packet. Whether we choose to deal with piracy like a system compromise, or through a separate judicial process, is a matter of policy. However, consider that the line between the two is not always so clear.

More comprehensive information on writing Snort rules can be found easily on the Internet. The following URL is the best place to start:
http://www.snort.org/docs/snort_manual/node14.html

Packet analysis on real-world cases

As mentioned above, reading packets from a known compromised host can help with rule generation. It can also help understand how a compromised system works to inform the attacker. The following are several packets culled from a live network. We should also look at a proposed rule or two that could better match the packets we are discussing, so we can understand how to adapt to new situations as they arise.

```
#(1 - 10) [2004-XX-XX 15:58:07]  Possible RogueIRC 06
IPv4: 139.102.X.X -> X.X.X.X
     hlen=5 TOS=0 dlen=166 ID=62152 flags=0 offset=0 TTL=127 chksum=7885
TCP:  port=3664 -> dport: 42844  flags=***AP*** seq=1788584791
     ack=1599969501 off=5 res=0 win=15304 urp=0 chksum=11289
Payload:  length = 126

000 : 50 52 49 56 4D 53 47 20 23 70 31 20 3A 5B 53 43   PRIVMSG #p1 :[SC
010 : 41 4E 5D 3A 20 52 61 6E 64 6F 6D 20 50 6F 72 74   AN]: Random Port
020 : 20 53 63 61 6E 20 73 74 61 72 74 65 64 20 6F 6E    Scan started on
030 : 20 31 33 39 2E 31 30 32 2E 78 2E 78 3A 34 34 35    139.102.x.x:445
040 : 20 77 69 74 68 20 61 20 64 65 6C 61 79 20 6F 66    with a delay of
050 : 20 31 30 20 73 65 63 6F 6E 64 73 20 66 6F 72 20    10 seconds for
060 : 30 20 6D 69 6E 75 74 65 73 20 75 73 69 6E 67 20   0 minutes using
070 : 31 30 30 20 74 68 72 65 61 64 73 2E 0D 0A         100 threads...
```

Our first packet is the result of another rule from the Educause security list. Like the RogueIRC 03 and 04 rules above, this rule looks for a PRIVMSG followed by another string, in this case “Scan”. The host IP addresses and times have been sanitized for security reasons. The machine on 139.102.X.X, the university network, is reporting to its master server in Beijing. The ASCII representation on the packet is kept on the right-hand column, hex on the left. This packet is reporting that the bot is beginning a port scan of TCP 445 against the university network; very likely it is looking for 'lsass' vulnerabilities. Note that the “139.102.x.x” is part of the original packet, not a sanitized result, so the attacker is quite aware of the correct range to scan. The bot is also apparently using 100 different scanning threads, each with a delay of 10 seconds in between scans. One theory behind the delay between scanning is to prevent detection by not consuming too many resources, either at the workstation or network level, to be noticed. Through this packet we can see that one machine with an IRC bot, or any sort of compromise in the enterprise, can quickly lead to large-scale scanning and exploitation. If the perimeter firewall normally blocks this scanning, it may not be blocking this attack because it occurs from one machine internal to the network to another. If that is not worrisome enough, consider that the packet above actually came from an unsecured wireless network within 139.102.X.X – we have no information as to who owns that machine!

```
#(1 - 1) [2004-XX-XX 15:50:26]  Possible RogueIRC 05
IPv4: 139.102.X.X -> X.X.X.X
     hlen=5 TOS=0 dlen=148 ID=4425 flags=0 offset=0 TTL=127 chksum=65452
TCP:  port=1038 -> dport: 42844  flags=***AP*** seq=130447279
     ack=2246909574 off=5 res=0 win=15497 urp=0 chksum=23413
Payload:  length = 108

000 : 50 52 49 56 4D 53 47 20 23 70 31 2D 73 63 61 6E   PRIVMSG #p1-scan
010 : 20 3A 5B 46 54 50 5D 3A 20 46 69 6C 65 20 74 72    :[FTP]: File tr
020 : 61 6E 73 66 65 72 20 63 6F 6D 70 6C 65 74 65 20   ansfer complete
030 : 74 6F 20 49 50 3A 20 31 33 39 2E 31 30 32 2E XX   to IP: 139.102.X
040 : XX XX 2E XX XX XX 20 28 43 3A 5C 57 49 4E 44 4F   XX.XXX (C:\WINDO
050 : 57 53 5C 53 79 73 74 65 6D 33 32 5C 72 65 61 6C   WS\System32\real
060 : 70 6C 61 79 2E 65 78 65 29 2E 0D 0A               play.exe)...
```

This packet is nice enough to report the actual destination of its attack, as well as the file on the destination machine that is part of the trojan being

installed. This is the direct result of the scan initiated above. These messages are returned as status messages to keep the attacker informed of the progress of each bot. Here we can see that the 139.102.X.X address (sanitized, this is a specific machine not a range) is being compromised from the scanner host. It is very likely that this new machine will also be set up to do scanning, perhaps for a different vulnerability. Although the packet reports FTP, this may actually be reporting TFTP traffic.

```
#(1 - 78) [2004-XX-XX 16:27:01]  Possible XDCC Activity [MOVIE]
IPv4: X.X.X.X -> 139.102.X.X
     hlen=5 TOS=0 dlen=172 ID=14550 flags=0 offset=0 TTL=50 chksum=40999
TCP:  port=6669 -> dport: 1919  flags=***AP*** seq=2580276337
     ack=424066451 off=5 res=0 win=6432 urp=0 chksum=23554
Payload:  length = 132

000 : 3A 5B 65 58 5D 58 44 43 43 2D 50 44 2D 39 39 34   :[eX]XDCC-PD-994
010 : 35 21 7E 65 58 40 38 33 31 31 34 30 46 2E 38 41   5!~eX@831140F.8A
020 : 35 38 44 41 37 32 2E 33 43 41 36 31 31 42 46 2E   58DA72.3CA611BF.
030 : 49 50 20 50 52 49 56 4D 53 47 20 23 65 78 74 72   IP PRIVMSG #extr
040 : 65 6D 65 2D 6D 6F 76 69 65 7A 20 3A 54 6F 74 61   eme-moviez :Tota
050 : 6C 20 4F 66 66 65 72 65 64 3A 20 31 31 33 36 2E   l Offered: 1136.
060 : 32 20 4D 42 20 20 54 6F 74 61 6C 20 54 72 61 6E   2 MB  Total Tran
070 : 73 66 65 72 72 65 64 3A 20 33 33 35 2E 33 35 20   sferred: 335.35
080 : 47 42 0D 0A                                       GB..
```

This packet is from an IRC bot on the external network, from a site which appears to be a co-location service in the United States. As seen in the first line of the alert, it was generated by the MOVIE rule from the previous section, and it demonstrates another use for IRC bots. This bot is sharing about 1G of movies on the #extreme-moviez channel -- not comparatively much, but we hope the customer is not paying per gigabyte transferred, for they have incurred over 335G in illicit traffic since this single IRC bot has been running. This activity can grow to the size of your network; if a single workstation is allowed a large chunk of bandwidth, it may start using all of it if the IRC bot node is not dealt with in the first few days of infection. In this case, the internal machine on 139.102.X.X is likely accessing this channel on purpose to download files illegally.

```
#(1 - 82090) [2004-XX-XX 09:50:32]  BLEEDING-EDGE IRC - Nick change on non-std port
IPv4: 139.102.X.X -> X.X.X.X
     hlen=5 TOS=0 dlen=96 ID=47031 flags=0 offset=0 TTL=127 chksum=21758
TCP:  port=2470 -> dport: 8888  flags=***AP*** seq=3334969967
     ack=4177268192 off=5 res=0 win=16560 urp=0 chksum=62008
Payload:  length = 56

000 : 4E 49 43 4B 20 5B 41 5D 2D 4D 65 64 61 42 6F 74   NICK [A]-MedaBot
010 : 73 2D 78 30 0A 55 53 45 52 20 70 77 6E 74 20 33   s-x0.USER pwnt 3
020 : 32 20 2E 20 3A 50 72 6F 70 65 72 74 79 20 6F 66   2 . :Property of
030 : 20 23 41 6E 69 6D 65 0A                            #Anime.
```

This is the result of one of the bleeding edge rules, designed to detect NICK changes on ports that are not normally associated with IRC. In practice, this kind of rule is very useful, since almost no legitimate IRC traffic takes place over these ports. The packet size is kept below 64 bytes in this rule (“dsize:<64” in snort) so as to not trigger a lot of false matching.

```
#(1 - 1519) [2004-XX-XX 11:50:28]  suspicious machine (incoming)
Ipv4: 4.23.X.X -> 139.102.X.X
 hlen=5 TOS=0 dlen=56 ID=14798 flags=0 offset=0 TTL=117 chksum=61516
TCP:  port=64284 -> dport: 6667  flags=***AP*** seq=4020794036
 ack=4142342231 off=5 res=0 win=64190 urp=0 chksum=45518
Payload:  length = 16

000 : 50 4F 4E 47 20 3A 31 43 33 42 38 33 30 32 0D 0A   PONG :1C3B8302..
```

This is the first packet we look at that was culled from a traffic sniff of

an IRC bot server located in a student dormitory. Around 500 hosts were connecting to this server. This incident has provided a few interesting packets, logged by Snort using a rule matching all traffic going to this particular IP. The above packet shows the PONG response sent by the client on a broadband connection to the server's PING query.

```
#(1 - 1502) [2004-XX-XX 11:50:28]  suspicious machine (incoming)
IPv4: X.X.X.X -> 139.102.X.X
      hlen=5 TOS=0 dlen=86 ID=15874 flags=0 offset=0 TTL=115 chksum=60922
TCP:  port=29575 -> dport: 6667  flags=***AP*** seq=823935648
      ack=1918099077 off=5 res=0 win=64512 urp=0 chksum=15138
Payload:  length = 46

000 : 4E 49 43 4B 20 61 6B 73 6E 77 73 6E 66 71 0D 0A   NICK aksnwsnfq..
010 : 55 53 45 52 20 6C 75 69 76 64 72 6D 63 20 30 20   USER luivdrmc 0
020 : 30 20 3A 61 6B 73 6E 77 73 6E 66 71 0D 0A         0 :aksnwsnfq..
```

This packet above shows a typical registration of a client, again from a broadband connection, most likely a home user. Because the client's NICK must be unique from other clients, a random string is usually used to register the IRC bot.

```
#(1 - 1503) [2004-XX-XX 11:50:28]  suspicious machine (outgoing)
IPv4: 139.102.X.X -> X.X.X.X
      hlen=5 TOS=0 dlen=120 ID=10621 flags=0 offset=0 TTL=127 chksum=63069
TCP:  port=6667 -> dport: 29575  flags=***AP*** seq=1918099077
      ack=823935694 off=5 res=0 win=16514 urp=0 chksum=46560
Payload:  length = 80

000 : 45 52 52 4F 52 20 3A 43 6C 6F 73 69 6E 67 20 4C   ERROR :Closing L
010 : 69 6E 6B 3A 20 5B XX 2E XX XX 2E XX XX 2E XX XX   ink: [X.XX.XX.XX
020 : XX 5D 20 28 54 6F 6F 20 6D 61 6E 79 20 75 6E 6B   X] (Too many unk
030 : 6E 6F 77 6E 20 63 6F 6E 6E 65 63 74 69 6F 6E 73   nown connections
040 : 20 66 72 6F 6D 20 79 6F 75 72 20 49 50 29 0D 0A    from your IP)..
```

This indicates that too many connections are taking place from one IP address to the bot server and that the server is limiting the amount of connections per IP. An educated guess would be that the IP address is the address of a home router, behind which several different machines are infected with the same IRC bot. This is another example of how the client-server and client-replication method used can get at machines behind a perimeter. If one machine behind the router is compromised (e.g. by a trojaned executable), it can recruit its neighbors behind the router if they are vulnerable to attack.

```
#(1 - 1715) [2004-XX-XX 11:50:31]  suspicious machine (incoming)
IPv4: X.X.X.X -> 139.102.X.X
      hlen=5 TOS=0 dlen=104 ID=22655 flags=0 offset=0 TTL=112 chksum=9646
TCP:  port=55441 -> dport: 6667  flags=***AP*** seq=2897706553
      ack=101749144 off=5 res=0 win=16130 urp=0 chksum=39661
Payload:  length = 64

000 : 50 52 49 56 4D 53 47 20 23 6D 65 73 73 61 67 65   PRIVMSG #message
010 : 73 23 20 3A 5B 6C 73 61 73 73 5F 34 34 35 5D 3A   s# :[lsass_445]:
020 : 20 45 78 70 6C 6F 69 74 69 6E 67 20 49 50 3A 20    Exploiting IP:
030 : 31 39 32 2E 31 36 38 2E 34 2E 32 32 39 2E 0D 0A   192.168.4.229...
```

Now we are at the other end of this type of packet from our earlier examination. This packet is being sent from the client IRC bot in Chile to our IRC botnet server in the dormitory, to report that it is attempting to compromise 192.168.4.229, a private network address which is not routable and therefore has not been sanitized. The client bot has found another neighbor to infect, probably one that would otherwise be thought secure.

```
#(1 - 2359) [2004-XX-XX 11:50:40]  suspicious machine (outgoing)
IPv4: 139.102.X.X -> X.X.X.X
      hlen=5 TOS=0 dlen=400 ID=13124 flags=0 offset=0 TTL=127 chksum=33901
TCP:  port=6667 -> dport: 4844  flags=***AP*** seq=144657477
```

```
      ack=2877783056 off=5 res=0 win=16258 urp=0 chksum=20154
Payload:  length = 360

000 : 3A 5B 6B 69 6C 6C 61 5D 2D 38 35 33 35 32 32 21   :[killa]-853522!
010 : 75 6C 64 71 69 79 75 40 36 43 32 43 33 30 2E 44   uldqiyu@6C2C30.D
020 : 46 42 41 34 45 44 32 2E 34 42 33 33 45 38 44 37   FBA4ED2.4B33E8D7
030 : 2E 49 50 20 4A 4F 49 4E 20 3A 23 6A 75 6C 69 65   .IP JOIN :#julie
040 : 23 0D 0A 3A 69 72 63 2E 6E 61 76 2E 63 6F 6D 20   #..:irc.nav.com
050 : 33 33 32 20 5B 6B 69 6C 6C 61 5D 2D 38 35 33 35   332 [killa]-8535
060 : 32 32 20 23 6A 75 6C 69 65 23 20 3A 2E 64 6F 77   22 #julie# :.dow
070 : 6E 6C 6F 61 64 20 68 74 74 70 3A 2F 2F 77 77 77   nload http://www
080 : 2E 74 72 65 6E 7A 68 6F 73 74 2E 63 6F 6D 2F 66   .trenzhost.com/f
090 : 69 6C 65 73 2F 64 6F 6E 64 6F 6E 2F 7A 61 6D 2E   iles/dondon/zam.
0a0 : 65 78 65 20 63 3A 5C 6F 68 6B 38 32 2E 65 78 65   exe c:\ohk82.exe
0b0 : 20 31 20 2D 73 0D 0A 3A 69 72 63 2E 6E 61 76 2E    1 -s..:irc.nav.
0c0 : 63 6F 6D 20 33 33 33 20 5B 6B 69 6C 6C 61 5D 2D   com 333 [killa]-
0d0 : 38 35 33 35 32 32 20 23 6A 75 6C 69 65 23 20 61   853522 #julie# a
0e0 : 73 20 31 30 39 35 31 37 33 31 39 38 0D 0A 3A 69   s 1095173198..:i
0f0 : 72 63 2E 6E 61 76 2E 63 6F 6D 20 33 35 33 20 5B   rc.nav.com 353 [
100 : 6B 69 6C 6C 61 5D 2D 38 35 33 35 32 32 20 40 20   killa]-853522 @
110 : 23 6A 75 6C 69 65 23 20 3A 5B 6B 69 6C 6C 61 5D   #julie# :[killa]
120 : 2D 38 35 33 35 32 32 20 0D 0A 3A 69 72 63 2E 6E   -853522 ..:irc.n
130 : 61 76 2E 63 6F 6D 20 33 36 36 20 5B 6B 69 6C 6C   av.com 366 [kill
140 : 61 5D 2D 38 35 33 35 32 32 20 23 6A 75 6C 69 65   a]-853522 #julie
150 : 23 20 3A 45 6E 64 20 6F 66 20 2F 4E 41 4D 45 53   # :End of /NAMES
160 : 20 6C 69 73 74 2E 0D 0A                            list...
```

Here we have a long packet from server to client, but we include it because it shows a few interesting things. Firstly, the [killa] probably refers to the Internet handle of the attacker. We also see a command to download the file “zan.exe” as “c:\ohk82.exe” from www.trenzhost.com. Since the system should already be trojaned, this is likely an update or additional software that will be run by the compromised hosts. The channel name also appears to be “#julie#”. Curiously, the 'trenzhost' site was not active at the time this packet was logged; perhaps it was a victim of a denial-of-service itself, because the ISP was notified or the IRC bot was too successful in propagating. It also would not be surprising if the bot kit contained its own 'hosts' file to direct this named traffic to another host instead of the typical DNS result. Only a packet analysis of the compromised machine itself can lead to proof.

```
#(1 - 10580) [2004-XX-XX 11:52:49]  suspicious machine (incoming)
IPv4: X.X.X.X -> 139.102.X.X
      hlen=5 TOS=0 dlen=142 ID=5086 flags=0 offset=0 TTL=111 chksum=36996
TCP:  port=47797 -> dport: 6667  flags=***AP*** seq=2477825822
      ack=1063236490 off=5 res=0 win=64231 urp=0 chksum=28856
Payload:  length = 102

000 : 50 52 49 56 4D 53 47 20 23 6D 65 73 73 61 67 65   PRIVMSG #message
010 : 73 23 20 3A 5B 46 54 50 5D 3A 20 46 69 6C 65 20   s# :[FTP]: File
020 : 74 72 61 6E 73 66 65 72 20 63 6F 6D 70 6C 65 74   transfer complet
030 : 65 20 74 6F 20 49 50 3A 20 31 30 2E 31 30 2E 38   e to IP: 10.10.8
040 : 2E 32 37 20 28 43 3A 5C 57 49 4E 4E 54 5C 53 79   .27 (C:\WINNT\Sy
050 : 73 74 65 6D 33 32 5C 77 6E 6D 70 6C 79 72 2E 65   stem32\wnmplyr.e
060 : 78 65 29 2E 0D 0A                                  xe)...
```

Here is another incident where a machine is compromising another over a private network (10.*). This packet is reporting back to the server that the IRC bot in Denmark has been successful in putting a trojan executable onto the target machine. The trojan executable is named similar to Windows Media Player, so the person at the workstation may think it is a normal process running.

```
#(1 - 13653) [2004-09-14 11:53:36]  suspicious machine (outgoing)
IPv4: 139.102.X.X -> X.X.X.X
      hlen=5 TOS=0 dlen=350 ID=47968 flags=0 offset=0 TTL=127 chksum=46758
TCP:  port=6667 -> dport: 3020  flags=***AP*** seq=3992909909
      ack=246474892 off=5 res=0 win=16300 urp=0 chksum=38436
Payload:  length = 310

000 : 3A 76 76 63 77 61 6D 61 68 21 6E 79 68 79 66 68   :vvcwamah!nyhyfh
010 : 77 40 44 69 65 2D 33 35 42 33 34 38 37 31 2E 70   w@Die-35B34871.p
020 : 6F 6F 6C 38 30 31 38 31 2E 69 6E 74 65 72 62 75   ool80181.interbu
```

```
030 : 73 69 6E 65 73 73 2E 69 74 20 4A 4F 49 4E 20 3A   siness.it JOIN :
040 : 23 6C 61 73 74 69 6D 65 23 0D 0A 3A 69 72 63 2E   #lastime#..:irc.
050 : 6E 61 76 2E 63 6F 6D 20 33 33 32 20 76 76 63 77   nav.com 332 vvcw
060 : 61 6D 61 68 20 23 6C 61 73 74 69 6D 65 23 20 3A   amah #lastime# :
070 : 2E 61 64 76 73 63 61 6E 20 6C 73 61 73 73 5F 34   .advscan lsass_4
080 : 34 35 20 31 35 30 20 33 20 39 39 39 20 2D 62 20   45 150 3 999 -b
090 : 2D 72 20 2D 73 0D 0A 3A 69 72 63 2E 6E 61 76 2E   -r -s..:irc.nav.
0a0 : 63 6F 6D 20 33 33 33 20 76 76 63 77 61 6D 61 68   com 333 vvcwamah
0b0 : 20 23 6C 61 73 74 69 6D 65 23 20 61 73 20 31 30    #lastime# as 10
0c0 : 39 35 30 36 32 35 30 35 0D 0A 3A 69 72 63 2E 6E   95062505..:irc.n
0d0 : 61 76 2E 63 6F 6D 20 33 35 33 20 76 76 63 77 61   av.com 353 vvcwa
0e0 : 6D 61 68 20 40 20 23 6C 61 73 74 69 6D 65 23 20   mah @ #lastime#
0f0 : 3A 76 76 63 77 61 6D 61 68 20 0D 0A 3A 69 72 63   :vvcwamah ..:irc
100 : 2E 6E 61 76 2E 63 6F 6D 20 33 36 36 20 76 76 63   .nav.com 366 vvc
110 : 77 61 6D 61 68 20 23 6C 61 73 74 69 6D 65 23 20   wamah #lastime#
120 : 3A 45 6E 64 20 6F 66 20 2F 4E 41 4D 45 53 20 6C   :End of /NAMES l
130 : 69 73 74 2E 0D 0A                                 ist...
```

Lastly, we see a command to do 'lsass' vulnerability scanning being given. The options to the scan (150, 3, 999) are likely the number of threads, seconds between scan, and minutes to scan respectively.

It is interesting to look at these packets, but what have we gained? Like many security problems, we need to be able to periodically review and refine our approach. Looking at these packets, we can see avenues for better rule generation.

One approach is to have each rule focus on a separate area of attack. First, let us propose a rule to spot any traffic matching 'lsass' on any port 6667, bidirectional. It would be necessary to replace 'lsass' by new vulnerabilities when they are released, which is why it is important to look at these packets of compromised machines periodically. By using this rule we are looking for the obvious weeds in the garden, and not trying too hard to do more with just one rule. By keeping the port range small, and the string to match specific (although short), we hope to not see many false positives. One disadvantage of short text strings is that they will occur by chance in a certain percentage of packets. The longer the text string, the easier it is to filter out this background traffic.

alert tcp any any -> any 6667 (msg:"IRC BOT 1 - lsass"; flow:to_server,established; content:"lsass"; nocase:; classtype:bad-unknown; sid:3011381; rev:1;)

Secondly, consider the bottlenecks in this process for the attacker. What steps must be accomplished, regardless of what port or kit the attacker uses? In all cases, the client must register a NICK request to identify themselves. How will we best match this rule, without generating a lot of nonsense alerts? Note that the NICK packets above contain an uppercase NICK at offset 0. By matching traffic on every port but the most common port, we can be assured that the machines we find are in fact compromised. This rule matches less packets, but finds better results, and works as a nice complement to rule #1. By setting an offset and depth, we hope to reduce the false positives that would otherwise occur with such a small textual string.

alert tcp any any -> any !6667 (msg:"IRC BOT 2 - NICK begins packet TCP !6667"; flow:to_server,established; content:"NICK"; offset:0; depth:5; classtype:bad-unknown; sid:3011382; rev:1;)

Third, we would like to see if things are happening on the network internally that a solely perimeter IDS might not detect. To accomplish this, we can use the IRC bot reporting information against the attacker by looking for

“File Transfer” followed by private range IP addresses, or the address of our internal network. These two rules show examples of this line on thinking, although the IP signature on rule 3a will need to be adjusted to fit other networks.

alert tcp $HOME_NET any -> any any (msg:"IRC BOT 3a - file trans internal IP"; flow:to_server,established; content:"file transfer"; nocase:; within:80; content:"139.102"; classtype:bad-unknown; sid:3011383; rev:1;)

alert tcp $HOME_NET any -> any any (msg:"IRC BOT 3b - file trans private IP"; flow:to_server,established; content:"file transfer"; nocase:; within:80; content:"192.168"; classtype:bad-unknown; sid:3011384; rev:1;)

Finally, we want to detect if any bot servers are being deployed on the network. Typically a machine with a large amount of outgoing connections will be flagged by network engineers, but at an organization such as a university, these incidents get buried under a mountain of other file-sharing traffic and exploitation. We would like rules specific to this problem. A generic rule, unlike rule #1, would be preferable. Rule #2 doesn't match the most common port, and rule #3, although reversible, depends upon the reporting information from the client. We will instead craft a rule based on a different component, the “PRIVMSG” message, incoming to our network. Case-sensitivity will save some processing time, and the text string is unique enough not to find many false positives. We can use the offset to further thin the detection.

alert tcp any any -> $HOME_NET any (msg:"IRC BOT 4 - possible internal BOT server"; flow:to_server,established; content:"PRIVMSG"; offset:0; depth:8; classtype:bad-unknown; sid:3011385; rev:1;)

Problems with this approach

There are several problems with using this approach. If the overall intent is to deal with system compromise and improve security in general, it is a difficult strategy to pursue. First, this method is reactive, not proactive. Machines will be compromised by the time the first IRC packet is seen. Rules of this sort are not always robust enough to feed into a script to disable machines automatically, so must be filtered by human hands.

Secondly, finding this activity presents a moving target. The customizable kits that carry this payload change on a daily basis. What today is sending constant pings to a machine in Sweden on TCP 6667 may tomorrow be slowly port scanning the network and connecting once a week to a server in Italy on TCP 43023. In 2001, one of the targets used to set up bot networks was a variant of CodeRed II compromised systems called “Powerbot” (Dittrich). Today, it is XP workstations vulnerable to ‘lsass’ exploits. In order to catch the most intrusion, generic rules must be used, which in turn causes so many alerts that security staff can not keep pace with reading and reacting to them. Even a basic rule looking for JOIN on TCP 6667 will catch many false positives, if a lot of legitimate IRC traffic takes place on the network -- anyone joining a standard IRC channel will trigger this

rule. It is the esoteric connections that are of interest, but those are by definition harder and more time-consumptive to find. Although a difficult task in theory to find every compromise, it is relatively easy to get decent results with a few simple rules such as the above. Hopefully our journey to writing some new rules, and looking at the results, has been a good learning experience.

Getting results with this approach

Using Snort to detect this activity allows the coordination of cleanup, the monitoring of evolving trends, and an evaluation of security within the network. A Snort rule which polls for IRC traffic matching a known exploitation signature can produce useful alerts. Those alerts can be selected from the alert database and mailed automatically to security personnel with a small amount of programming ability. As systemic intrusion grows, the ability to contain the problem must grow with it. Dealing with each individual machine by-hand may not be a sustainable way to deal with the problem.

If a certain area of the network has a much higher incidence per machine of IRC bot intrusion, such as a student dormitory, we can be sure the security of that zone is rather poor and machines in that area must suffer constant attacks. This should indicate that it is an area that needs improvement. It also will be a good area to monitor more closely using Snort, perhaps with a separate sniffer with a pared down rule set for monitoring limited kinds of activity. The more specific and 'local' the IDS can be, the more useful this information can become. Scanning and exploitation on a computer lab could lead to some sort of programmatic action to be taken at the network level. These open zones are likely the place within the network where new exploits will first be tried, and by examining this data, what is learned can be leveraged to predict problems in the rest of the enterprise.

Concluding statements

A neat, packaged solution to this problem does not exist without strict control of workstations. Since strict control is not always possible or practical, and the Internet enables attackers to download a new set of tools at 4:00AM and deploy them on our network at 4:03AM, dealing with mutating problems like IRC bots will be a constant battle. Intrusion detection systems are a must-have in the current security environment. However, having the tools is a meaningless gesture without the time and ability to use them properly. This paper has looked at one small part of the problem, that of IRC bot compromise. We have seen insight into IDS rule writing and deployment throughout the enterprise. The focus of this paper dealt mainly with a University environment, because that is the author's area of study. Open enterprises, along with ISPs, can also be considered 'enablers' of this activity, for security issues often are ignored, deemed unimportant, or otherwise not given due consideration at an organizational level. It is important that organizations such as these be proactive in dealing with security issues not only for their own good, but for the good of the Internet as a whole. Even if the reader has no intention of writing rules, deploying IDS systems, or analyzing security practice, it is hoped that at least an understanding of one the recent problems of computer security has been advanced.

References

Dittrich, Dave. "'Power' bot". August 8, 2001. URL: http://staff.washington.edu/dittrich/misc/power.analysis.txt (7 Oct 2004).

Esler, Joel. "Bleeding Snort rules". 2004. URL: http://www.bleedingsnort.com/bleeding-all.rules (7 Oct 2004).

Graham, Robert. "On Magic, IRC wars, and DDoS." URL: http://www.robertgraham.com/op-ed/magic-ddos.html (7 Oct 2004).

Holst, Alex. "Frequently Asked Questions about the Dancer bot." June 17, 1998. URL: http://dancer.sourceforge.net/FAQ (7 Oct 2004).

Holstein, Michael. "[unisog] Computer Pests (known trojan/virus list)". September 29, 2004. Posting to Educause security mailing list (re-post of rules). URL: http://lists.sans.org/pipermail/unisog/2004-September/015232.html (7 Oct 2004)

Oikarinen, Jarkko and Reed, Darren. "RFC 1459 – Internet Relay Chat Protocol." May 1993. URL: http://www.faqs.org/rfcs/rfc1459.html (7 Oct 2004).

Rayome, Jerry. "IRC on Your Dime? What You Really Need to Know About Internet Relay Chat". June 1998. URL: http://ciac.llnl.gov/ciac/documents/CIAC-2318_IRC_On_Your_Dime.pdf (7 Oct 2004).

Roberts, Paul. "ISP Telenor cripples zombie PC network." September 10, 2004. http://www.computerworld.com/securitytopics/security/cybercrime/story/0,10801,95847,00.html (7 Oct 2004).

Stenberg, Daniel. "History of IRC (Internet Relay Chat)." September 24, 2002. URL: http://daniel.haxx.se/irchistory.html (7 Oct 2004).

Wagner, Cliff. "The Edge Zone: IRC Zone: IRC Bots." 2002. URL: http://www.edge-zone.net/irc/bots.html (7 Oct 2004).