Global Information Assurance Certification Paper

An Analysis of a Root Compromise

Rockie Brockway

I. Introduction

This paper will recount and analyze a root compromise that occurred at an ISP in 1999. The majority of reports of root level break-ins are found to be direct results of vulnerabilities in existing code, such as buffer overflows. Buffer overflows occur in the process of allocating memory when there is more data than the buffer (such as an array) was coded to contain. As the overflowed data backtracks up the stack and overwrites other elements such as additional local variables and the frame pointer, it eventually reaches the return address on the stack that points to the called subroutine's main program. When executed properly, the overflowed data then places a new return location onto the stack, which points to the actual address in memory where the attacker's blackhat (e.g. bad) code would reside[1]. From that point forward the system should be considered broken into and taken over.

Another form of root compromise that shows up quite often, but doesn't really get as much of the limelight as the infamous buffer overflow, occurs in the form of plain old sloppy coding. Poorly written cgi-bin scripts are good examples of such code that, when called, do not check bounds of input variables or content, are running suid, and for purposes of this analysis, do not check permissions on files, among other poor coding practices.

The following paper focuses on the background, details, and events of one such root compromise that I was involved in very directly, and will analyze it according to the recommended GIAC Incident Handling steps of Preparation, Identification, Containment, Eradication, Recovery and Follow-up. It will serve to represent how NOT to handle an intrusion incident and our follow-up/lessons learned section will be dense. Additionally it will serve to make a point that, although security awareness and implementation is on the rise, much of the industry still views security as a separate entity that can be added on top of existing infrastructure that will just work. The reality is that security is an architectural issue, which should be an element of infrastructure design from the moment the initial business plan and company direction has been completed, if possible.

II. Background

In 1991 an associate and friend of mine converted his existing Bulletin Board System over to a dialup Internet Service Provider with the addition to a dedicated 56K DS0 pulled to his very cold basement. It became the first public ISP in the region and I became its first employee. From 1997 to 2001 I was the senior systems and network administrator, being lead for both the systems group who handled all of the servers and day to day internal situations and the network group, who were responsible for both the internal network operations and external customer installs. Additionally I took on the role as Security Administrator over the years. In 1995, as System Administrator, I began a

series of automated scripts for user management written in PERL, which gave the Tech Support staff easy access and modification rights to user information both on the Solaris based user's home machine and the tac_plus based dialup authentication system. In 1997, discussions between myself and the company's CEO regarding converting the entire operation to one global SQL database in the backend lead us to research many immature market available solutions, none of which would do everything we wanted to accomplish in the undertaking. We had wanted to combine all of our billing, AAA and tech support databases into one main database which in theory would solve the fact that the current system of multiple databases were never synced up. The fact that the simple action of placing a user on hold for non payment meant entering this in the billing database, the Solaris passwd file, the tac_plus user file and in the tech support trouble ticketing system was unacceptable.

III. The Project, 1997-1999

After many months of researching inferior programs and of me telling our CEO "I do not wish to design and implement this software project myself," I in fact was told to design and implement this software project myself, which, of course, I did. However, we were nowhere near being ready to migrate all of the data into one main database. So in the meantime I set my goal to design a series of programs that would at least have one main control center (a web-based user interface) that from there could go out to each of the important boxes and execute the appropriate scripts that would complete each particular event. The design was relatively simple – write some RPC servers that would sit out on the appropriate boxes and run the appropriate scripts depending on the variables sent to them from the client main user interface safely nestled on the inside staff network. The boxes were all running secure RPCBind, which uses libwrap to perform host-based client control and were only allowing RPC connections from our inside staff machine running the web-based user interface. The RPC client/server[2] software was encrypting the data communications between each machine. The entire suite of programs was coded in PERL, since at the time I had been developing the majority of the system scripts with PERL. Many precautions regarding the script's user input were thought of and handled, including running each server in PERL's TAINT mode[3]. This forced the servers to un-TAINT each variable passed to them by the client user interface through a filter of acceptable characters, as well as bounds checking in both the user interface code and the server code to ensure that hopefully buffers could not be overflowed.

The current authentication system, the freeware tac_plus, designed but unsupported by Cisco Systems, could not easily make the transition to authenticating via an SQL database. In the few months of research, everyone had the feeling that we should move everything into an Oracle database, despite the fact that nobody in our organization had any real experience with it. So the decision was made to migrate to Cisco Secure ACS, which could not only use a database backend but also support both tacacs and radius[4]. So after a few months setting up, configuring and migrating data for the new database backed authentication system, I continued working on my suite of programs for central user control.

Once everything was battle tested and ready we installed and converted everything at once, which under the circumstances went very well. My staff and I had prepared well for the event, at this point over a year in the making. The new system was up and running and my suite of programs was doing its job very well.

IV. Preparation

As noted above, I had attempted to ensure that my code was relatively secure by checking input parameters and using Perl's TAINT mode. The servers in our network were all Sun Sparc based machines running the Solaris 2.6 operating system at the time the software was created. My staff and I were fairly diligent in maintaining the latest patches and watching bugtraq for any security information relevant to our own systems. Aside for the main user machine, which is where all of our user's home directories were located, each machine had inetd.conf stripped to the bone, had tcpwrappers running for sshd connections from our internal network, had secure RPCbind running and only served one main purpose. I had convinced the CEO a couple of years back that safety in numbers combined with many small machines each doing only one thing was a solid system design. So the farm was MANY Sparc 20s and Ultra 1s, each running smtp, tac_plus, bind or nntp, etc. Most of the servers had secondaries for failover in the event of an outage due to a crash or upgrade.

The main user machine was another story, one with which I never felt comfortable. But the difficulties involved in making the changes necessary to lessen the potential dangers, especially for our tech support staff, which would have to deal with the fallout of an extended outage during the changeover, led to the "Ignorance is Bliss" downward spiral. This user machine ran sendmail for mailbox distribution, ftp for anonymous file retrieval and user web pages, pop3, imap4, ssh, and telnet, as well as many shells which were given to users upon request (but not by default). Due to the fact that at the time most users had never even heard of ssh and that there was no decent windows port, the remote people telnetting in from other networks to access their shells were regularly getting their passwords sniffed. My staff and I were used to this and kept a keen eye on who was logged in from where and investigated as many irregularities as we could. This led to many users being placed on hold until we could contact them, explain that we really didn't believe they were in Seattle, San Juan, and Helsinki in the same week, and change their password for them. Npasswd, a better passwd change program[5], was installed with strict rules on this box so the users with shell access could not change their password to anything that did not conform to our password policy. The rest of the non-shell users had passwords given to them by the account creation staff, all of who had been trained to make hard to guess, complex passwords.

Most of the servers on top of it all were logging everything to an internal syslog box on the staff network, as well as their own log files. Tripwire had been installed on most boxes, but was really only taken seriously on the user machine. Even then, it got ignored for months at a time. Full backups of the machines ran once a week using ufsdump. This was not the best procedure, but it was enough to recover from lost files or accidental user deletions a few times. Like most organizations, ours suffered from too much work and

not enough people or hours in a day. We had excellent intentions and did our best to not only run a well-oiled ISP that had an excellent reputation of connectivity, uptime, and support, but make it as secure as we could as well. One of the biggest problems was that despite all of the above, there was no written security policy except the one in my head that I had handed down in training to each of my staff. There was only a user's Acceptable Usage Policy, which stated what the remote users themselves could and could not do on our system.

Above the host layer of security I had both a distribution layer and a perimeter layer of security as well. The distribution layer consisted of access-list filters per VLAN on our core Cisco 5500 layer 3 switch. Most of the individual services in our network had their own VLAN (e.g. news, mail, authentication), so I had configured these access-lists to only allow incoming traffic destined for only the appropriate ports associated with the services on that VLAN. Further out at our edge we had multiple core Cisco 7200 routers with multiple T3s connecting us to our uplink distributors. Each of those high-speed interfaces were also configured to restrict both inbound and outbound traffic. Inbound restrictions included any traffic which had a source of any network owned by us, as well as services such as SNMP, SMB, RPCbind/portmapper and X11 that I felt had no reason entering our autonomous system. Outbound traffic was being filtered for egress spoofing, so none of our subscribers could engage in any kind of spoofing activity.

V. Identification

SU 06/09 14:28 + pts/12 userx-root

Any system administrator will tell you that the above line found in Solaris' /var/adm/sulog directly results in sinking, black-hole type feelings full of despair, failure, curiosity and anger. It says that someone who shouldn't have the root password to the machine does indeed have the root password to the machine. My staff was gathered and briefed that we had an intruder who somehow had gotten root access. Assignments were handed out and we went to work gathering as much forensic evidence as we could. One thing we could unfortunately not do was bring the box down. We had 15,000 dialup users dependent on the one box that had no secondary failover, and an unannounced outage to gather evidence of how someone had been able to get root access on our box was out of the question.

A quick look at the /var/adm directory showed us the following:

```
drwxrwxr-x   9      root    sys          512 Jun  7 03:10 .
drwxr-xr-x   33     root    sys          512 May  8 09:11 ..
drwxrwxr-x   5      adm     adm          512 Sep  1 1999 acct
-rw-------   1      uucp    bin          0 Sep  1  1999 aculog
drwxr-xr-x   2      adm     adm          512 Sep  6  1999 exacct
-r--r--r—    1      root    root         216384 Jun  15 14:58 lastlog
drwxr-xr-x   2      adm     adm          512 Sep  1  1999 log
-rw-------   1      root    sys          0 Oct 23 1999 loginlog
```

```
-rw-r--r--   1   root   other   938319 Jun 15 07:54 messages
-rw-r--r—    1   root   other   1355045 Jun  11 03:09 messages.0
-rw-r--r--   1   root   other   2005252 Jun  4 03:06 messages.1
-rw-r--r--   1   root   other   1978713 May 28 03:09 messages.2
-rw-r--r--   1   root   other   1197996 May 21 03:07 messages.3
drwxr-xr-x   2   adm    adm     512 Sep  1  1999 passwd
drwxrwxr-x   2   adm    sys     512 Jun  15 00:00 sa
drwxr-xr-x   2   root   sys     512 Sep  1  1999 sm.bin
-rw-rw-rw-   1   root   bin     0 Sep  1  1999 spellhist
drwxr-xr-x   2   root   sys     512 Sep  1  1999 streams
-rw-------   1   root   root    5250 Jun  15 14:13 sulog
-rw-r--r--   1   root   bin     8556 Jun  15 14:58 utmpx
-rw-r--r--   1   root   root    970 Apr  20 11:01 vold.log
-rw-r--r--   1   root   other   70250 Jun 15 14:58 wtmp
-rw-r--r--   1   root   adm     182652 Jun 15 14:58 wtmpx
```

The first thing noticed is that for this system, the lastlog, wtmp and wtmpx files, which contain the user information that gets returned to programs such as last, and who and w (for an entire history of who, when and where accounts have logged in), were exceptionally small. We were used to seeing the wtmpx file well over a few hundred MB before getting around to rotating it out. Not one member of the staff recalls doing such a thing recently, so it looked like our guest may have completely removed the original lastlog and wtmp files to hide his tracks.

We started our identification analysis with the account that we know got root access, userx.

last userx

```
userx    pts/0     as3-22.my-company.net   Thu Jun 15 08:46 - 15:05  (06:19)
userx    pts/8     JCVLA010-0574.sp   Thu Jun 15 08:43 - 08:45  (00:02)
userx    pts/0     JCVLA010-0261.sp   Tue Jun 13 17:41 - 18:12  (00:31)
userx    pts/24    as4-13.my-company.net   Tue Jun 13 15:14 - 17:23  (02:09)
userx    pts/40    w250.z216112023.   Tue Jun 13 15:11 - 15:14  (00:02)
userx    pts/5     w250.z216112023.   Tue Jun 13 14:18 - 15:11  (00:53)
userx    pts/37    w250.z216112023.   Tue Jun 13 14:14 - 14:18  (00:03)
userx    pts/5     w250.z216112023.   Tue Jun 13 12:24 - 14:14  (01:50)
userx    pts/18    w250.z216112023.   Tue Jun 13 11:03 - 12:24  (01:20)
userx    pts/36    w250.z216112023.   Tue Jun 13 09:11 - 11:03  (01:51)
userx    pts/25    w250.z216112023.   Tue Jun 13 09:06 - 09:11  (00:05)
userx    pts/22    as1-42.my-company.net   Mon Jun 12 14:58 - 18:22  (03:23)
userx    pts/11    JCVLA010-0304.sp   Mon Jun 12 11:55 - 13:49  (01:54)
userx    pts/16    JCVLA010-0317.sp   Mon Jun 12 11:44 - 11:52  (00:08)
userx    pts/29    JCVLA010-0501.sp   Mon Jun 12 11:25 - 11:44  (00:18)
userx    pts/21    PPPa57-ResaleJac   Mon Jun 12 11:18 - 11:21  (00:02)
userx    pts/24    PPPa57-ResaleJac   Mon Jun 12 11:09 - 11:17  (00:08)
```

```
userx    pts/5     PPPa57-ResaleJac   Sun Jun 11 16:31 - 16:31  (00:00)
userx    pts/5     PPPa66-ResaleJac   Sun Jun 11 16:28 - 16:29  (00:00)
userx    pts/36    as3-13.my-company.net   Fri Jun  9 14:55 - 09:11 (3+18:15)
userx    pts/22    nas-86-195.atlanta.n  Fri Jun  9 14:54 - 14:55  (00:01)
userx    pts/20    nas-86-195.atlanta.n  Fri Jun  9 14:24 - 14:54  (00:29)
```

We can see that within the span of 6 days we had userx coming into us from many locations. And in between (even crossing over in one case) we had what was probably userx's real account dialing into our company's access servers. The program last, operating on the Solaris OS, cuts off the Fully Qualified Domain Name if it is longer than it likes, but it was not a problem since we had telnet wrapped with tcpwrappers, and that syslog output of who was logging in was being sent to a different syslog server on our internal network. But we will not expand the offending hosts for sanitation purposes.

Further searches on the existing lastlog file with the domain names gave us a list of user accounts that had been logging in from similar hosts, and that list totaled out to nine other users. But the attacker already had the original root password and access, as noted by the fact that the sulog clearly showed they successfully used it to gain root access. Assuming that the attacker had already downloaded the /etc/shadow file, it would be trivial to run Crack against the hashes and eventually get more passwords. The root password, which obviously at this point had been changed by us, wasn't even safe, since we still had no idea how the attacker had gotten it in the first place, nor were we certain that the attacker had not left any backdoor into the machine. The fact that the password had been changed in the first place should have been a good indication that they had been found out.

VI. Containment

We took our list of users whom we suspected had compromised accounts and placed them all on hold, so at least the ones we thought we knew were cracked could not log back in to try to do it all over again. We also modified tcpwrappers to deny any traffic from the known or suspected domains that these cracked accounts had logged in from. The Solaris OS was set up to NOT allow remote logins by root[6]. If the point of entry was a cracked user account instead of a software vulnerability that we had missed a patch for, we were hoping that we had a decent shot at keeping them out while we continued our assessment. Not a good thing to rely on, but it was all we had at the moment.

Although we still were nowhere near in a safe spot, we needed to assess the damage. Tripwire had been installed and neglected, but did shed some light on what programs had been modified. Modified ps, ls, netstat, w, who, rm, mkdir and du were discovered by tripwire and verified by comparing MD5 hashes with known good binaries of the same patch level. Tripwire at the time was not being run automatically and the database had not been updated in a couple of months, so I wanted to double check.

The old and reliable tiger audit program was used to search the system for anomalous signs, such as odd named or hidden directories, UID 0 accounts, etc. Nothing was found from that report. Known safe versions of netstat and lsof combined found nothing

listening on an odd or unknown port that shouldn't have been there, so we at least did not have an unknown server setup on the box. Nothing had been added to the root crontabs, and the programs called from that crontab had not been modified, so we were ruling out any timed trojans.

Userx's .bash_history had not been wiped and there was evidence that eggdrop, a common IRC bot installed on hacked systems, had been downloaded and compiled, and then placed in userx's /home/www directory. Although none of my staff nor I was a seasoned incident handling expert, it was becoming clear that this may not have been the work of an expert hacker, but more likely a script kiddie.

Ttywatcher, a program that allows for the monitoring and logging of user's sessions, was installed. We had then compiled a modified version of bash that logged every command executed at its shell to syslog, which in turn logged out to our internal syslog server. In the event that our intruder(s) got back in we at least could follow their steps. Process accounting was turned on. The domains from which each potential hacked account came were denied by tcpwrappers. We were beginning to feel more secure but were seeing login drops in the wrapper logs so we knew they were still out there.

VII. Eradication

One of my staff sent me mail a couple of days into the containment process asking me why there would be a file called /etc/shadow.b1. I replied by saying that it was a temporary file created by one of the scripts called by this machine's RPC server that put user's on and off hold status. We achieved this in a VERY rudimentary way. If a user either violated our AUP or was not paying their bills, the script would flock (file lock) the shadow file, make a copy, insert or remove a * in front of the user's password hash and copy the new version of the shadow file back to /etc/shadow. I was then informed that this copy was left in /etc with 644 permissions (i.e. world readable). I felt immediately we had found our hole. I also felt immediately sick that I had been so sloppy as to leave a world readable shadow password file around on the filesystem.

I rapidly fixed the coding error by first ensuring that the proper umask was set up and that any files created by the scripts would always have permissions 600, and second by removing the copy as soon as it was no longer needed. A full code revision would follow in a couple of weeks.

I next gave the word to keep the last known good backup of the filesystems prior to our incident handy and to back up the entire filesystem once more to DLT via ufsdump (patched). We were going to not only nuke the box from high orbit, but also upgrade it at the same time to Solaris 7. I felt that under the circumstances and given the situation, a quick and fresh start was the best solution, now that we felt we understood and corrected the original problem.

A great amount of coordination took place in the next couple of days. We sketched out exactly what needed to get reinstalled on the new system and in what order, which new

programs we would have around to help us in an event like this in the future, and what was to be logged remotely. Plus, the tech support staff was instructed to take the newly revised holduser program and place every user with a shell account (roughly 300) on hold. This forced users to call up and have tech support change their passwords, upon proper verification that the person calling was indeed that user.

VIII. Recovery

The new system came up with relatively little problems. We had applied the latest Recommended and Security Patches for Solaris 7 and began installing all of the necessary additional software that our users were used to having access to. All of the additional software was at the latest version, freshly downloaded. Some new versions of existing software that were compiled had some problems, but the overall upgrade and cleanup was a success. Every shell user had their passwords changed and warned not to use telnet (although to this day the box still accepts telnet connections which is something I had fought). Prior to final network connectivity, I scanned the box with nessus and saint, and then ran tripwire, moving the database to a different machine for safekeeping. Snort was installed to monitor the host itself for intrusion attempts. Finally, the code for our internal suite of RPC servers was gone over with a fine-tooth comb for any further coding mistakes. None were found.

Over the next month, weekly scans of the box were performed on top of daily monitoring of all logs (including tripwire and snort) on the user machine. We saw very little within the normal noise of a system this size that threw up any warning flags regarding any additional attempts to regain unauthorized access to this box.

IX. Follow Up/Lessons Learned

After going over much of the details with the members of my staff at that time, the biggest improvement necessary is LOGGING! The majority of this account, although accurate, was by memory and discussion. Very little at the time was logged, not even to paper. This entire event is an example of Incident Handling by the inexperienced. If anything regarding this event ever happened to make it to court it would have been tossed out due to lack of evidence. We were very lucky to have been attacked by what were probably kids who sniffed some user passwords and got into our machines, then stumbled upon the pot of gold, the /etc/shadow backup file with world-readable permissions. They did not know enough to clean out anything but the basic log files, which were more than likely out of a root kit they downloaded somewhere.

There were a vast number of things wrong with our preparation:

1) No Security Policy or warning banners existed
2) There was no specific Incident Handling Team
3) Logs were not being monitored on any kind of regular basis
4) Tripwire was virtually ignored
5) There was no policy for auditing internal code

6) Backups of the system were only run once a week (full backup vs. incremental or differential), and were never tested and verified to be good. We had been lucky.

Additionally, once the event was confirmed and identified to be an incident, no single person lead was assigned to the incident, no detailed action logs were made, and little logs of intruder activity were kept. Authorities were never notified. No evidence was recovered, tagged and labeled. If the incident ever made it to a court of law, there would have been nothing concrete to prove our systems had been breached. And we did a disservice to other networks involved in where the intruders were coming from by not reporting the fact that a break-in had occurred that was sourced from their networks.

Our containment of the incident scored only slightly better than our preparation and identification. We kept a relatively low profile, backed up the systems, blocked the networks we felt were participating in the attack, and placed the known or suspected cracked accounts on hold. Given the fact that this particular machine had no secondary, the decision was made to keep it online during the process of investigation and damage assessment. The backup made was a simple ufsdump –0cu of all the filesystems, not a dd or bit copy which could have served as either legal evidence or a complete duplicate to dissect offline.

Once the code mistake was located in the hold/unholduser portion of the scripts, we felt we could focus on the eradication and recovery processes. We had a good feeling that we knew how the system had been compromised and the extent of the damages, so we took the opportunity to not only clean the system, but upgrade the OS as well. The new Solaris 7 operating system was installed, the OS was brought up to the latest patch levels, and all extra programs the users were used to having were recompiled with the latest versions. All users who had shell access were placed on hold and forced to change their passwords. Obviously the root passwords were changed as well. The users' home directories were then restored using the last backup of the system. We felt confident that there were no odd or hidden directories, no unusual SUID programs, etc., to restore these home directories from the backup made after the incident. A tripwire database was generated once we felt we had most of the system prepared to go back online. And this time we vowed to pay attention to it.

A couple of days prior to the reinstallation of the system, I reported to our CEO that there had been a breakin, but that we felt we had it under control and that we would be reinstalling and upgrading the machine in question to be sure. The response was more or less "Oh, OK. That's interesting, but good work." And that was the extent of my after incident report, aside from jotting down a couple of details like dates and a few screenshots that had been saved. Clearly, my follow-up procedures were essentially non-existent and needed attention.

X. Summary

This account is a very good example of poor incident handling. However, I learned enough from this incident to go out and find the information required to successfully

prepare to deal with an incident (like this course). I had been doing focused security work for years, but felt what I knew was incomplete. I had many strengths (host security, network perimeter and router security, firewalls, virus technology, and network intrusion detection) but lacked in overall incident handling procedures, code checking, and policy knowledge. This incident not only illustrated to me what these obvious weaknesses were, but also highlighted some not so obvious weaknesses as well. The latter could be blamed on either laziness or the too many responsibilities excuse, but include the lack of consistent log monitoring and feeling secure for the simple fact that some piece of software had been installed, even though it was not being checked regularly (e.g. tripwire).

I had developed my own layered security model for this organization that started out at the edge routers. Ingress and egress spoof filtering, denial of inbound services like SNMP, RPC, SMB and NFS from entering our autonomous system, and ensuring the directed broadcasting was disabled on every interface[7]. Our inner distribution layer 3 switch was filtering traffic and only allowed, for example, tcp 49 (tacacs) from our access servers and tcp 22 (ssh) from our internal office network to our authentication servers. The servers themselves had been stripped down so they were only running the necessary services for whichever purpose they had and snort was running on each of them for insurance (which was actually monitored fairly regularly via Snortsnarf).

The main user machine was indeed the wildcard of the bunch, and consequently was the only server that was ever hit with an incident. Aside from host based filtering via tcpwrappers and a regular run of PatchReport, there was little else done to this machine due to its very nature – it was an open user machine. Even so, those facts proved to be only insignificant in relation to this incident, whose core vulnerability was poor coding of system programs. It was only a matter of time before someone figured that out. Given the lack of remote access policy which still allowed users of this machine to telnet and ftp into it from remote systems, password sniffing was a fact of life and we grew accustom of seeing an account here or there logging in from different places. When one was discovered, the account was turned off until the paying owner of said account was notified and given the news that their password seems to have been sniffed during some previous telnet/ftp/pop session and to use ssh in the future where possible. Their password was then changed and the filesystems searched for any files/directories owned by that account, Tiger was run to search for anomalies, Nessus was run to scan for new open ports and potential new vulnerabilities and process accounting logs and history files gone over in case we missed anything.

The entire incident illuminates the dangers of not really visualizing the scope of a company's direction as soon as possible. Not thinking about merging multiple databases into one until a few years into the life of the organization hurt severely from a secure architecture point of view. The company then decided even later to bring that project in house to an already small and busy administrator team, which was then pressured to get it finished. Security, although thought of in the software's design, was not as important as getting the product completed. All of this eventually led to a large security incident, which itself was poorly handled.

The above situation occurs time and time again in small to mid-sized organizations: Internally written programs are rarely checked for any type of poor coding practices. That combined with the equally frequent scenario of an undermanned staff needing to take on more responsibility than they should handle, thereby thinning out the level of focus on detail, begins to open holes in the corporate security model (if a true model even exists). The lack of any sort of coherent policy outlining not only your backup and incident handling procedures but the structured hierarchy and responsibility assignments simply makes the situation even more difficult to handle if an incident actually occurs. And the probability that an incident will occur is getting greater each day with the vast number of automated scan/attack tools readily available for download to anyone who can use a browser.

Today, security for any organization that has any type of network connected to the Internet is finally being widely considered. It is reaching the point where many companies are making proactive decisions rather than reactive ones. Many companies are getting over thoughts like "we've never had an incident before, we must be secure" and preparing themselves, their staff, and their networks for the worst. Paranoia is now a good thing. However, security services still seem to be considered an additional item placed on top of existing infrastructure rather than part of a whole solution. Integration of security pieces into existing networks is one thing, but being able to successfully take an organization already rooted in their ways and be able to not only handle existing security weakness but also apply a new policy based on the business model and existing architecture is quite another. Having the ability (let alone the opportunity) to get down to the nitty-gritty well beyond the run of the mill "when are your backups run" and "is your firewall configured properly" to "are we sure that this person responsible for x, y and z is not spread out too thin?" Even if your staff is composed of the greatest administrators in the world, if they have too much that they are responsible for, things will invariably fall through the cracks and be missed by them.

This leads to another potential layer of the layered security model - the layer of well-trained management. It is critical for upper management to be aware of exactly how much load their people can safely handle and offload responsibility when necessary. Combining good perimeter and host defenses with a good policy that ensures code checking, backup schedules and test restores, log monitoring, vulnerability assessments, and incident procedures, as well as other items specific to the company and their business model, greatly reduces the chance of incidents. The lack of any or all of the above can potentially be disastrous. Vision that encompasses the entirety of a company's goals is crucial in implementing a secure architecture from the ground up. And when security from the ground up or beginning is not possible, it is the responsibility of the security staff or security consultants to ensure these points are not overlooked and that security is not just thrown at existing architectures in the assumption that it will just work on top of everything already built.

XI. References

[1] Oh, Taeho. "Advanced Buffer Overflow Exploit." Oct 21, 1999. URL: http://packetstorm.securify.com/papers/unix/adv.overflow.paper.txt.

[2] Weidmann, Jochen. "RPC::pServer - Perl extension for writing pRPC servers." URL: http://www.perldoc.com/cpan/RPC/pServer.html.

[3] Birznieks, Gunther. "CGI/Perl Taint Mode FAQ." Version 1.0. June 3, 1998. URL: http://www.gunther.web66.com/FAQS/taintmode.htm.

[4] Cisco Systems. "CiscoSecure ACS 2.3 for UNIX Reference Guide." URL: http://www.cisco.com/univercd/cc/td/doc/product/access/acs_soft/cs_unx/csu23rg/index.htm.

[5] Hoover, Clyde. "Npasswd - A better password change program." Version 1.2. July 13, 1999. URL: http://www.utexas.edu/cc/unix/software/npasswd/.

[6] Sabernet. "Solaris Security Guide." June 23, 1999. URL: http://www.sabernet.net/papers/Solaris.html.

[7] Cisco Systems. "Essential IOS Features Every ISP Should Consider." Version 2.6.5. Dec 4, 1998. URL: http://www.cisco.com/public/cons/isp/documents/IOSEssentialsPDF.zip.