Global Information Assurance Certification Paper

Reducing the Risk associated with Authentication and Authorization through the deployment of SUDO and Powerbroker: A Case Study in Information Security

Steve Mancini

Version 1.4b
October 14, 2003

God grant me the serenity
to accept the things I cannot change;
courage to change the things I can;
and wisdom to know the difference.
--Reinhold Niebuhr[1]

Abstract:

Take a walk in my shoes. You are the security manager for a UNIX environment that is composed of several different flavors of UNIX which totals several thousand nodes. You have 5000+ customers, both local and remote, who are all entitled to varying degrees of access to information that ranges up to and including company secrets regarding future technologies. Prior to your assuming the role of security manager, these customers could request "resource accounts" which were used in their business processes. You have 50+ system administrator peers who all currently have access to the super-user account, root. Some of these administrators are using sudo, some are not. You see a need to put a bit more control over this authentication/authorization process. You attend a SANS course which only confirms the course of action you are planning to address the weaknesses in your authentication and authorization scheme. Like most companies, your finances are not were they were a few years ago during the height of the dot com boom, so your choices need to not only make sense from an information security and usability perspective, but also must be a sound financial decision.

You have 2 options currently available to you which can improve the situation – one is the popular freeware tool sudo (http://www.courtesan.com/sudo). With it you can allow people to authenticate as other users and provide a degree of access control by specifying what commands they can run on what hosts. The alternative is Powerbroker, by Symark (http://www.symark.com). Powerbroker provides a slightly superior level of granular control, plus the additional benefits of keystroke logging when desired. This case study will explore each of these options, their strengths and weaknesses as they apply to a large scale work environments and their implications in considering your authentication - authorization process, and will offer up one possible solution which uses both applications in a manner to minimize some of the risks known to exist with shared accounts, both traditional and super-user.

[1] While I am not a religious person, I have often found myself reflecting upon this wisdom and recognized it is completely appropriate to anyone responsible for securing a production environment.

"Before"

Like most professional environments, my task to clean up and refine our authentication and authorization program would need to balance the needs of my users to be productive against my goal of reducing risks associated with their (and our own) processes. I am using the At this point I was sufficiently educated in the proper approaches to deploying a security solution – so I knew I needed to understand the work model and the policies which I would seek to enforce before I dove into technical solutions. Many of the roles and work models that I would have to account for are sufficiently universal that my approach should be applicable to other organizations inside my parent corporation (and perhaps externally).

The security posture of the corporate environment could best be described as a collection of fiefdoms with relative autonomy to conduct business as they see fit so long as the local policies did not incur a risk to the corporation as a whole. My fiefdom spans 2 US states. All of the perimeter security between my networks and the "wild" are maintained by other orgs in IT – we have very limited direct exposure to the internet and the threats associated with it. My fiefdom reports into a larger organization which works in parallel with the corporate IT efforts. The charter of my local organization is to empower our users and provide specialized support that the corporate IT program was unwilling (and unable) to offer. We possess our own NIS domains, NFS servers, etc. Our efforts in the windows world are in sync with the corporate directive (and since they are outside the scope of my efforts in this project, this is the first and last mention of them for the sake of providing a complete picture of the environment in which I operate). Corporate policy relies upon a few well known security principles:

- principle of least privilege
- need to know access rules
- separation of powers
- demonstrated business need to deviate from any policy or standard

These principles guide the decision I make. Like many work environments, technology decisions are akin to Pandora's Box - once opened, it would require a tremendous effort to take away the process/privilege/access the users have been given. As such, I need to recognize that my customers have grown accustomed to the possession of accounts which they use to conduct their work. Some have grown dependent upon the ability to have super-user access on specific machines or for specific file systems by generating automated processes which are essential for managing the mundane and routine tasks (chmod, chgrp, chown) that would otherwise impede progress on more important pursuits.

Recognizing the Risks:

I opted not to perform a formalized risk assessment on this situation – I was not confident there was sufficient ROI on such an effort. I understood from the beginning what I could hope to achieve and what was beyond attaining at this time. That said, there were still several key vulnerabilities (which I am confident are not unique to my business model and as such worth mentioning) identified that I needed to either address or at least be aware of for this project to be considered successful.

- A *lack of a coherent policy* existed regarding the use of resource accounts and which technology (sudo, powerbroker, suid programs) was used for which reasons. The lack of a coherent policy results in different administrators selecting different courses of action to meet our customer's needs. This weakens our overall security posture because all of these different solutions are often piecemeal and undocumented. This jeopardizes any premise of computing environment awareness that I, as the security program manager, might have. Peers could establish suid programs and/or create accounts which we as an organization may lose track of. Also, without a policy in place to refer to, few customer groups would think to change the password when there was employee turn-over. Members of the projects would leave for other jobs inside the company and still be able to access the old UNIX project accounts because no one would consider it necessary to change the passwords.
- S*uid programs* were the defacto solution often chosen to allow customers to execute single commands as root. Most of these revolved around the management of project files that resided in NFS. While well written suid programs and scripts may not pose a significant risk to an environment, those which are not well written can wreck extreme havoc. Likewise, there was an underlying assumption by many in my organization that our customers would never seek to use the suid programs written for any purpose other than the legitimate business needs that required their existence. As the security manager, I am paid to be paranoid so in evaluating the presence of these scripts and programs, I was not willing to accept the benevolence of my customers as a forgone conclusion.
- *Insecure password sharing* was considered the norm among the customer groups. In the case of the support group, the root password is stored in an encrypted file and all members of support are trained to understand how to decrypt the file in a secure manner. The customers, on the other hand, have no such training or the motivation to adhere to such rules. Early on I found several disturbing trends (which we joked about in the SANS class) that all sum up to really poor authentication management. The processes used to share passwords were very easy to intercept or socially engineer. Attacks against the confidentiality of the passwords would be too numerous to mention given the implemented practices not only of storing and communicating these passwords, but also in their creation and alteration. Passwords were shared on web pages (which the authors presumed we

either secured or so obscure no one would find them), sent out via unencrypted email, or in the worst cases – written on white boards in the lab!

- *Weak passwords* were often chosen for these shared accounts. Customers would select trivial-to-guess passwords because so many people had to remember it they would shy away from corporate standards of strong, complex passwords. The passwords most often were related in some way to the project's name or the organizational unit's identity. In a few cases, it was the worst possible password (short of no password) – it was the name of the account. Use of the password cracking tool John the Ripper would yield approximately 30-50% success on the resource accounts that were present in the environment. Often times these were also easily guessed so that a password cracker was not necessary; guessing the password was the account name would yield a few positive results.
- *Password aging* (the procedure by which users are required to periodically change their passwords over a given span of time) was not implemented because of a 'business restriction'. The shared passwords would not be changed at the required intervals. Worse, they would not be changed when membership tied to the resource account would change – so someone could switch groups or leave the company and the password would not be changed for fear of "disrupting critical path work." Without password aging in place, even reasonable strong passwords could be decrypted via a tool such as john the ripper. The lack of password aging also resulted in some accounts circumventing the advances in password strength that were required for general accounts. Even though my organization implemented new tools that would validate that a new password met certain requirements, these older passwords, which never needed to be reset, we established without the requirements and since they were never changed, their strength was never challenged through such processes.
- *Bypassing authentication* – early on in my analysis of the environment I discovered that almost every resource account had at least 1 user who could not (or would not) remember the password. So their solution was to add entries to the account's .rhosts file and bypass all needs to authenticate to gain access. Luckily we had processes in place to examine files for the most dreaded + sign and remove it and 'educate' the offending party (and their manager) of the threat this posed. Regardless, the risk still existed that someone intent on obtaining access could forge the right credentials to gain access to the account via these .rhost files, or they could circumvent the authorization process by adding people to these files who could then access the accounts without the password. In a large scale environment, especially one using open-ended ip address assignment technologies such as DHCP, it would be trivial for someone to bring up a system, create the accounts they wished to have access to, then simply rsh or ssh onto a production system and completely bypass any authentication challenge.
- No *accountability or logging* was in place to monitor who used the various accounts. Those with the password could become root and execute commands without any trace of their actions on any of the thousands of

systems in the environment. While sudo was in place for our administrators, most would (out of habit) still su to root. While the lack of sufficient logging in the environment was not in and of itself a vulnerability, the situation aggravates the security posture of the organization by promoting a myopic view of the environment. Attacks that are underway go un-noticed, and those that have already occurred leave a poor forensics trail through which one would hope to determine how the environment was compromised and what was done once they obtained access.[2]

- No *Separation of Duties* had been established to distinguish job roles among the various administrators. We were providing all members of the support group (50+ people) the root password which was usable on all UNIX systems in the environment regardless of their job function. This risk was augmented mid-development by the introduction of student interns into our operations and customer support center. The manager of this center established the expectation that these students would be given privileged access once they were properly trained. This gap in process results in little to no ability to provide for conduct accountability or for a mature defense in depth strategy. Providing all administrators equal access to all systems dilutes the ability to track actions on systems to specific users or at least a reduced subset of users. In the nightmarish case of a disgruntled employee, it leaves the entire infrastructure at the mercy of the employee's wrath. A more mundane and plausible scenario, it opens the entire environment up to accidental damage through poorly executed scripts designed to affect a range of systems.
- *Session Hijacking* opportunities were plentiful in the environment. Administrators did not comprehend the risk associated with leaving root shells open on UNIX systems for extended periods of time – many were unaware of the advances in this area of attack. These shells would sometimes persist for days (or longer) making them attractive targets to internal attacks. Advances in session hijacking software have opened up this method of attack to script kiddies. Lingering, often neglected shells on UNIX systems are prime targets for such conduct. Root shells are routinely left open for extended periods of time by administrators – these shells are "lost" among the several other sessions that often exist in the window managers used by the administrators.

I was provided the opportunity to attend Track 1 near the completion of this project. Even though I have been engaged in information security for a few years, I selected track 1 to round out my foundation in security, fill in some gaps in an otherwise reactively-developed security education, and to use it as the groundwork for studying for the CISSP. I was pleased to discover that the security principles advocated in the class re-enforced my chosen course of action which was nearing the completion of their initial implementation. As part of preparing this paper, I revisited the risks mentioned above and applied the general formula to each (see table below). Rather than use High-Moderate-Low, I

[2] I personally maintain a rather low confidence in the actual detection and capture of a attack in progress, but that is another story for another time (or another paper).

elected to assign simplistic values in their place. I found that this helped to better highlight the change in security posture. The numbers do not mean anything except as a mapping: High=3, Moderate=2, and Low=1. This confirmed my feelings with regard to the importance each held in my desire to secure the environment.

Table 1: Relative Risk "Before"

Issue	Threat x	Vulnerability	= Risk
A lack of a coherent policy	2	1	1
Suid programs	3	2	6
Insecure password sharing	3	2	6
Weak passwords	3	3	9
Password aging	3	2	6
Bypassing authentication	2	2	4
No accountability	2	2	4
No logging	2	2	4
No Separation of Powers	2	1	2
Session Hijacking	1	2	2

"During"

Documenting Requirements

As the security manager for the group, I inherited the 2 different tools we used to enable authentication (Powerbroker and sudo). Each was a few versions behind the current and I was tasked with updating each of them. As such, I decided to also take this opportunity to examine them, and evaluate whether we still needed both, or if we could potentially select 1 of them and retire the other. Any formalized (and fair) evaluation of competing products should start with a objective set of standards by which they would be judged – a requirements document. Looking back upon the risks that currently existed in my environment, the tool that would be selected would need to address the most important identified risks.

The technology selected must:

1. Provide for a ruleset with a granularity level that will support the guiding principles of the corporate security program (Principle of Least Privilege, Separation of Powers, Need to Know, and Exceptions based upon Business Need).
2. Reduce/remove our dependency on suid programs by providing either a replacement for the suid bit, or by providing a taxonomy that will allow for controlled commands to be issued (as root) by authorized people in accordance with specific criteria.
3. Reduce/remove the need for password sharing by allowing customers to either authenticate to another account using their user account credentials, or allow them to run commands as another user once they authenticate as their user account.
4. Allow customers to authenticate using their own user-account passwords which have mandatory content requirements that make them harder to crack.[3] This would also help reduce the temptation to bypass authentication.
5. Allow for the revocation of access to these accounts without impacting processes currently underway using that account. This in turn would allow for the enabling of password aging since the resource accounts would no longer be impacted by the changing of the authentication credentials.
6. Provide for sufficient logging to track access to the accounts.
7. Provide keystroke logging when designated in a specific rule.
8. Provide the ability to timeout a session when designated in a specific rule to reduce the likelihood of neglected session hijacking.
9. Provide for secure communication of any authentication credentials that could be solicited as part of the program.

[3] Customers are told to synchronize their UNIX and Windows account passwords to allow for samba authentication. Active Directory has been established with stronger password rules than those initially established for UNIX.

Option #1: sudo (1.6.7p5)

Sudo allows a system administrator to establish rule based access to super-user privileges on a UNIX system. Deployment of this tool requires the installation of an suid binary (sudo) on the systems and the establishment of a configuration file (sudoers) that is used to define permissible actions. There are a great number of articles already present on the internet about sudo[4]. Most of them make a passing reference to its implementation as part of hardening of an operating system. Sudo's popularity has developed such that many UNIX operating systems come with sudo installed as part of the operating system image. For those situations where this is not the case, installing sudo has also been thoroughly documented, both at sudo's main website (http://www.courtesan.com/sudo) and even in a well-written paper for SANS by Robert D'Agnolo[5]. It follows the very simple steps most often associated with UNIX freeware:

1. Obtain copies from a trusted website.
2. Verify the source code via the publicly available signatures and/or checksums
3. unpack the compressed source code
4. configure[6]
5. make
6. make install
7. modify the configuration file to reflect your organizations policy positions

Configure Options

While I prefer to pass over some of the more routine aspects of installation, there are a few configuration options that are worth mentioning at this point since they contribute to the enhancement of the security of the environment should. While not all of the below contributed to my evaluation of sudo, a few were critical to meeting my requirements; specifically those that would promote the use of secure communication channels (Kerberos). Several of these capabilities were either already in my environment, or were being considered as part of the growing defense in depth strategy:

- ✓ One Time Passwords is supported for both s/key and opie.
- ✓ SecurID
- ✓ Kerberos IV and V.
- ✓ PAM (Red Hat 5.x+, Solaris 2.6+, HPUX 11.0+)
- ✓ AFS (Andrew File System)
- ✓ Shadow passwords

[4] http://www.google.com/search?hl=en&ie=UTF-8&oe=UTF-8&q=sudo
[5] D'Agnolo, Robert, "Implementing SUDO to replace SU: A Case Study Also Involving NIS and RDIST", SANS paper for GSEC certification, SANS 2003
[6] For those who need a good laugh, I would advise you to review the "insult" options that you can configure into sudo. Just make sure the people in your environment are thick-skinned.

The configuration of sudo also allows the tool builder to define several options which were far more important in my pursuit of a secure alternative to the current environment. Sudo provides for logging – not only of failed attempts but also for successfully authenticated commands. Choices made at compile time allowe me to designate which facility and priorities would be used for both failed and successful attempts. I was also able to select to log to syslog or a file (or both). I opted for syslog because at this point in my investigation, I had already established a central log server and accompanying log analysis scripts – so these configuration options would augment that project by allowing me to roll the results of the sudo activity in my environment into the log analysis scripts.

I selected the option to be immediately notified of failed attempts via email so that I could more readily monitor my environment for suspicious activity. The final productized version sends email to an internal mail list which currently contains the email address for my pager as well as my email address. This allows for me to be notified after normal business hours (a time often thought ripe for incidents to happen). The mail list is modified as necessary to delegate responsibility to anyone in my support group should I be on vacation, or should I move on to another position[7].

Another configuration option which proved to be of value for me was `–with-timeout=NUMBER`. When a user successfully authenticates via sudo, they are issued a ticket which allows them to issue subsequent sudo commands without being challenged for a password (from that shell). The default for this is 5 minutes. To promote usage of sudo over su in my environment, I opted to increase this time from 5 to 30 minutes. While I fully concede this increases the risk associated with the use of sudo, this was one of those instances where I was required to balance productivity and compliance against security. I perceived this to be a (relatively speaking) small concession if it would result in more people using sudo than su. My hope was that by encouraging the use of sudo, the timed tickets would result in administrators who would use sudo to conduct their work and at the end of the day there would be less root shells left overnight which could be potential victims of session hijacking. The end result would be numerous small windows of opportunity rather than several instances open for long periods of time which could be hijacked opportunistically.

The Sudoers File

Sudoers, the configuration file for sudo, provides for several options that would directly help reduce the previously enumerated risks in the environment. The complete man page for the sudoers file can be found both in the source tarball as well as on the website for sudo[8]. This configuration file allows for you to determine who-what-where: who is allowed to issue what commands, on what machines. Extended Backus-Naur Form (EBNF) is used to define the grammars

[7] If you know of anyone in need of say, a homeland security director, drop me a line ☺

[8] http://www.courtesan.com/sudo/man/sudoers.html

used in the sudoers file. This is a very simple to understand format that lends itself to easy adoption. Since my company would beat me with a frozen carp and hide the body if I were to reveal the details of our configuration file, I can only point you to a "real life" example of a sudoers file which Alek Komarnitsky has graciously put out on the net[9]. While this could look intimidating at first, once you understand the EBNF metasyntax, it is actually rather easy to work with. The 4 aliases defined (User, RunAs, Host, and Cmnd) made it possible to set up complicated rules which matched the policy positions that would be adopted as part of this project. As a result most entries in my sudoers file followed the pattern:

```
User_Alias     Host_Alias = (Runas_Alias) Cmnd_Alias
```

User_Alias	rule defining who could run sudo for a given grammar
Host_Alias	the systems they would be allowed to execute sudo on
Cmnd_Alias	the commands they would run
Runas_Alias	who the commands would be run as

The User_Alias provides the ability to define a user by their username, their uid, or their inclusion in a UNIX group or a netgroup. The Host_Alias can be defined by ip address, system name, netgroups, or CIDR notation. By modifying the Host_Alias entries, we could entitle or revoke access to a sudo account without having to impact all of the remaining users (such as you would if you suddenly needed to change the password for an account). By coupling User and Host aliases, I was able to overcome the issue of a separation of powers. Now, administrators could be designated to have the most basic "root privilege" based who they were, what groups they were in, and thus be given access to specifically designated machines, or categories of systems. In one case I was able to assist a lab owner with the delegation of super-user privilege to a select few people on a specific subnet which landed entirely within his lab by using CIDR notation.

The introduction of the Runas_Alias expanded my ability to use sudo for resource accounts. This would allow for my customers to use sudo to access the resource accounts either on a per-command basis, or they could use sudo and open a shell to the resource account via a command such as:

```
/path/to/sudo –u  RESOURCE_ACCOUNT /bin/tcsh
```

Using this command, my customers would be prompted for the password for their own user accounts. The passwords for the shared account were summarily disabled (replaced with a simple X in the password field thereby removing any chance of the account having its password cracked).

[9] http://www.komar.org/pres/sudo/sudoer15.html

The Cmnd_Alias expanded my ability to empower our customers on a granular level by allowing me to establish only specified commands which they would be permitted to run. This capability removed the need for suid scripts in my environment – by designating the users, the systems it could be run on, setting runas to root, and then detailing the exact path to, the command, and the parameters, I was able to provide root run scripts for my customers. Not only did this empower my customers, it also allowed me to track usage because each issuance of the command (and who issued it) was also logged.

Having deployed the new version of sudo in a test area of our production environment, it was time to assess sudo against the requirements:

Table 2: Risk reduction scorecard for sudo

Requirment	Score[10]
Rule sets promoting Principle of Least Privilege and Separation of Powers	1.0
Remove need for password sharing	1.0
Authentication by Knowledge tied to user accounts	1.0
Revocation of access without impacting other users	1.0
Logging to track access times and commands	1.0
Session keystroke logging	0.0
Session Timeouts	0.5[11]
Secure communication of authentication credentials	1.0
Acceptable Cost to deploy environment wide	1.0
Total	6.5/9.0

Drawbacks to SUDO:

While sudo met many of the requirements that I generated at the outset of this evaluation, I did also need to take into account the negative factors and consider if any of these would introduce a greater risk into the environment than those that this tool would help mitigate.

One of my first concerns was the potential to abuse sudo through issuing a command that would result in a shell being opened for root. To do so would circumvent many of the benefits sudo provided: (1) the commands subsequently issued to the shell command would not be logged, (2) an open shell would not honor the timeout functionality and could leave the account open for as long as idled[12] permitted on the system, (3) open shells would not be affected if we

[10] Scoring: 0 = fail, 0.5 = partial reduction, 1.0 addresses the risk

[11] While sudo lacks an absolute session timeout, the use of tickets provided a similar functionality that would likewise contribute to the reduction in opportunities to hijack sessions.

[12] "Idled wakes up at regular intervals and scans the system's utmp file to see which users are currently logged in, how long they have been idle, whether they are logged in more than once, etc. Idled then warns and logs out users based on a set of rules in its configuration file." – man idled(8) on RH linux

revoked access to the account. While it is possible to establish exception lists (so we could say that the Cmnd_Alias is anything except specific shells) it is trivial for the determined user to circumvent this by copying their favorite shell to a command they could use, then calling it.

A second concern that was raised was that the potential remained initially for the customers to backdoor their way from acceptable hosts (as defined by the Host_Alias) to those they are not authorized to access with the sudo commands or accounts. Through the use of a .rhosts (or .shosts) they could expand the scope of systems they could use the resource account on or allow others to access the account without using sudo. This was addressed through 2 steps which would hinder such conduct. First, we locked down the ~/.*hosts files for resource accounts:

```
% rm ~resource_acct/.rhosts
% mkdir ~resource_account/.rhosts
% /bin/chown root ~resource_acct/.rhosts
% /bin/chmod 000 ~resource_acct/.rhosts
% (repeat for ~resource_acct/.shosts)
%
```

The second step was already in place for alternate reasons but was modified to reinforce our policy decision. As part of our efforts to remove the use of + signs from the environment, we created a cron job which periodically examines home directories for compliance with several policies (permissions set accordingly, no + signs in dot files, alerts of the presence of other usernames in the .rhosts file, etc). We modified this script so that it would also verify that the above changes remained in place (that .rhosts was a directory owned by root, etc).

Another concern that existed when I performed my evaluation was the prevalence (or lack) of vulnerabilities that existed in the code. While I have consigned myself as most that patching is now a routine component to my defense strategy, I did not want to sustain the use of applications which were routinely requiring security patches. I performed a search of the Common Vulnerabilities and Exposures database for sudo[13] and was relieved to discover that there were only 7 entries related to sudo; none released since 2002. Also, 2 of these were for operating systems not present in my environment.

Table 3: sudo vulnerabilies as reported by Mitre

Reference	CVE Description:
CVE-2002-0184	Heap-based buffer overflow in sudo before 1.6.6 may allow local users to gain root privileges via special characters in the -p (prompt) argument, which are not properly expanded.

[13] http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=sudo

CVE-2002-0043	sudo 1.6.0 through 1.6.3p7 does not properly clear the environment before calling the mail program, which could allow local users to gain root privileges by modifying environment variables and changing how the mail program is invoked.
CVE-2001-0279	Buffer overflow in sudo earlier than 1.6.3p6 allows local users to gain root privileges.
CVE-2001-1240	The default configuration of sudo in Engarde Secure Linux 1.0.1 allows any user in the admin group to run certain commands that could be leveraged to gain full root access.
CAN-2001-1169	keyinit in S/Key does not require authentication to initialize a one-time password sequence, which allows an attacker who has gained privileges to a user account to create new one-time passwords for use in other activities that may use S/Key authentication, such as sudo.
CVE-1999-0958	sudo 1.5.x allows local users to execute arbitrary commands via a .. (dot dot) attack.
CAN-1999-1496	Sudo 1.5 in Debian Linux 2.1 and Red Hat 6.0 allows local users to determine the existence of arbitrary files by attempting to execute the target filename as a program, which generates a different error message when the file does not exist.

Option #2: Powerbroker (2.8.2)

Introduction:
Powerbroker was already deployed for specific uses in the environment when I assumed the role of security program manager; if it wasn't already present, I doubt I would have considered its deployment[14]. Luckily, an earlier version had already been deployed so it was "in the running" when I was considering authentication/authorization strategies. Like sudo, Powerbroker provides for the granular assignment of authentication and authorization permissions to users based upon a pre-established rule set. Powerbroker, however, provides a far more robust rule set and other benefits which made it hard for me to completely dismiss this tool. Powerbroker is a client server architecture;. users issue commands from licensed clients which communicate with the master server. This server validates the credentials of the user through a password challenge. It compares the command to be issued against a rule set. If the command passes, it is executed. Failures are sent to syslog. In this manner, the central server becomes the authenticating and authorizing agent. You can set up redundant servers and provide for a fail-over priority.

At this point of my investigation, we already had customers using this tool for a few resource accounts on a very select set of systems. This is where my

[14] This is absolutely *not* meant as an insult to Symark or their product. I merely come from a very freeware oriented background.

previous comment about Pandora's Box becomes especially relevant. We also had developed some in-house scripts which were used to allow the customers the ability to manage NFS directories on a filesystem level (chmod, chown, chgrp). Furthermore, our customers had developed "business processes" around the presence of this tool[15]. The tool had already demonstrated its versatility. The biggest issue that remained at this point was the price. Symark provides a per node license scheme at a cost that made it prohibitively expensive for me to justify deploying this tool across the entire environment. What I needed to resolve was if the price was worth maintaining this tool in the environment.

Like sudo, Powerbroker provided for my concern that the communication channels used were secure. During installation you can choose to have communication between the clients and the servers encrypted with 3DES[16]. Also available during the installation process is the ability to set up powerbroker to work with a Public Key Infrastructure (PKI) using Secure Socket Layers (SSL). I have not taken full advantage yet of the PKI/SSL model but anticipate its adoption in the foreseeable future. The product also supports LDAP to query user information, which is a plus for any environment which is using LDAP to authenticate UNIX logins against a windows directory service.

The configuration file for powerbroker was not as easy as that of sudo. This is because the granular control that is possible with powerbroker far exceeds that of sudo. Their C like configuration files provide for all of the options possible with sudo – they define users, hosts that they can run on, commands they can run, and who those commands will be executed as. Powerbroker requires the same password authentication as sudo; a user must provide the password for their account. Powerbroker exceeded sudo's capabilities by providing for the following additional rule set capabilities:

- The ability to create separate policy files to address subsets of your overall policy. These are incorporated into the master configuration file as "include" statements. So you could have a policy configuration file for interns (interns.conf), and should someone in the UNIX group 'intern' call the powerbroker command, it would reference the intern.conf file. (ie, if `(intern) { "include intern.conf"}`)
- You can define which host the command will be executed on via the *runhost* variable.
- You can define periods of time during which operations are permitted or denied via the *timebetween()* and *dayname()* variables.
- The triggering of a mail alert regardless of success or failure of the command executed.
- The ability to produce interactive scripts which step-by-step prompt the user for specific information.

[15] Recall my prior comment early on about Pandora's Box – this is an example of it.
[16] This feature could also be enabled post-installation by adjusting the configuration files associated with the clients and servers.

- The use of a *runtimeout* variable provides for a maximum run time. While this could be set to infinite, its use provided for the ability to shut down any process related to a powerbroker session after a given amount of time.
- The ability to use if() statements and logical operators to generate cases to reflect policy.

As is apparent at this point, the configuration capabilities of powerbroker enabled me to surpass the Separation of Powers and Principle of Least Privilege with greater granularity than sudo.

Perhaps the biggest sell for the retention of Powerbroker was its ability to engage keystroke logging for specified rules and the ability to playback those logged sessions through a root-only command, pbreplay. This capability was crucial for use by both my customers and my peer group. Certain customer processes introduced the use of resource accounts into their critical path for development, during which time it could become necessary to understand every action taken on mission critical files. For our own use, we use this command for members of support who are "less trusted" either due to their status in the group (contractors, interns, etc) or simply because they are new to our extremely complex environment and we wish to be able to troubleshoot any problems that they may run into while operating as a privileged user.

Drawbacks of Powerbroker

As already mentioned, the cost of powerbroker made it prohibitively expensive for me to deploy it to my 5000+ systems, let alone across the larger organization. This was especially true during a period of time when management had set for the challenge to "do more with less". To its credit, I have been unable to locate any published vulnerabilities or exploits against powerbroker.

Table 4: Risk Reduction Scorecard for Powerbroker

Risk	Score
Rule sets promoting Principle of Least Privilege and Separation of Powers	1.0
Remove need for password sharing	1.0
Authentication by Knowledge tied to user accounts	1.0
Revocation of access without impacting other users	1.0
Logging to track access times and commands	1.0
Session keystroke logging	1.0
Session Timeouts	1.0
Secure communication of authentication credentials	1.0
Acceptable Cost to deploy environment wide	0.0
Total	7/9.0

Decision-making Process

As is seen by the above scorecards, neither solution would address all of my requirements on its own. Both scored reasonably close to each other (sudo = 6.5, powerbroker = 7.0). Hence there was no clear cut solution to my problem. My decision would need to be based upon my initial requirements as they addressed the risks I had identified in my initial assessment.

I. *Provide for a rule set with a granularity level that will support the guiding principles of the corporate security program (Principle of Least Privilege, Separation of Powers, Need to Know, and Exceptions based upon Business Need*).

While the rule set provided by Powerbroker clearly provides a superior language and granularity, sudo's grammars are sufficient to meet my needs in specific designated work cases where the absence of powerbroker's granularity was an acceptable risk. Sudo's rules meets our organization's needs when it came to trusted system administrators and resource accounts which are not part of critical path or otherwise requiring the assignment of a heightened security posture for their use. The ability to restrict access based upon day and time makes Powerbroker the right choice for enabling interns and contractors who should only be working under supervised conditions during normal business hours.

II. *Reduce/remove our dependency on suid programs by providing either a replacement for the suid bit, or by providing a taxonomy that will allow for specific commands to be issued by specific people in accordance with specified criteria.*

The use of either program provides for the removal of the suid bit from programs created in house to meet the needs of our customers. In each case these commands are now issued appended to the respective command:

```
/usr/local/bin/sudo [command]
/usr/local/bin/pbrun [command]
```

III. *Reduce/remove the need for password sharing by allowing customers to either authenticate to another account using their user account credentials, or allow them to run commands as another user once they authenticate as their user account.*

Both tools allow for the reduction in the number of shared passwords that are present in the environment. With the exception of a few resource accounts which require console logins, the use of either program will enable the removal of passwords which would otherwise be shared among all those who are entitled to access the account.

IV. *Allow customers to authenticate using their own user-account passwords which have mandatory content requirements that make them harder to crack.*

In each case, the user's Authentication by Knowledge is based upon a challenge against their current userid. While there are justifiable "work models" which would prevent the changing of a resource account password, no such justification is accepted for user accounts, which must adhere to stronger password rules.

V. *Allow for the revocation of access to these accounts without impacting processes currently underway using that account. This in turn would allow for the enabling of password aging since the resource accounts would no longer be impacted by the changing of the authentication credentials.*

For each solution, revocation is controlled through the configuration files. This allows for the removal of a user without the need to reset any form of "shared secret" that they might have been privy to. Removal of their userid from the configuration files prevents subsequent logins, but alone does not terminate any currently open sessions. This weakness is partially addressed by powerbroker in the absolute timeframe associated with its *runtimeout()* function.

VI. *Provide for sufficient logging to track access to the accounts.*

Both powerbroker and sudo provides for a greater knowledge of privileged access because each provided for logging of the information to syslog. Each could be configured to send relevant information to syslog which could be incorporated into the log analysis efforts already in existence.

Both tools would allow for almost immediate notification when someone failed to access these. In each case the ability to send email could be incorporated into a mail alias which sends a notification text message to my pager 24x7 alerting me of these failures.

VII. *Provide keystroke logging when designated in a specific rule.*

Only powerbroker provides keystroke logging which could be incorporated into the rule set as needed. This keystroke logging allows for review of commands executed in a session and has proven invaluable when tracking down the source of accidents in the environment.

VIII. *Provide the ability to timeout a session when designated in a specific rule.*

Only powerbroker provided the resources to establish absolute timeouts. While this attribute is desirable when dealing with a potentially hostile revocation of rights, it became more of an impediment than benefit when customers processes would be cut off without warning. As a result of the "business need", the

runtimeout() on some resource accounts was set to time spans so large that it practically negated the benefit otherwise acquired. Sudo's ticket policy proved sufficient to reduce the risk associated with lingering shells only so long as the users respected the "no shell" policy.

IX. Provide for secure communication of any authentication credentials that could be solicited as part of the program.

Each program provided sufficient encryption to meet the corporate requirements for the communication involved. The SANS course confirmed my prior beliefs that a switched network does not provide any immunity from the sniffing of passwords off the wire, it only raises the bar (a bit). Neither program was perceived to be a security risk because of the use of insecure communication channels.

Chosen Course of Action

Unlike many other projects where I have had to trade off some requirement(s) when introducing a new technology or policy into the environment, I was fortunate in that each tool provided several advantages with regard to my defense-in-depth strategy. My final deployment strategy would recognize that each tool was a value-add to the environment and as such each should be retained to address very specific needs in the environment. Deployment of Powerbroker would be a targeted deployment so as to minimize the need to purchase licenses that might go to waste unused on some nodes. Sudo would be retained to address the mass-scale need for authentication by the customer support group and be deployed to all clients and servers in the environment at little cost to the company.

Now that each would remain in the environment to address specific needs, I felt the first step was to clearly define the division-level usage models for each tool so that moving forward we would have a coherent, objective application of the tools which would justify the deployment strategy I had selected. In preparing the documentation for the department, I referred back to the Track 1 information regarding policy.[17]

In an effort to reduce risks associated with authentication and authorization, all members of support will adhere to the following guidelines[18]:

1. All new members of support and those who were contracted or co-ops are required to use powerbroker for super-user actions in the environment. This would insure support of a keystroke record of their actions. New members of support would be transitioned to sudo use once their manager

[17] SANS Security Essentials, Chapter 8

[18] The guidelines provided have been sterilized of organizational and corporate references, hence some of the rules may appear a bit vague or unclear.

felt that the engineer had sufficient experience and exposure to the environment that we could “take off the training wheels”.

2. The root password would only be distributed to members of support whose role and responsibilities required them to login at the console. All others would only be given sudo access. In case of some emergency, specific members of our support group had access to a PGP encrypted file which contained the password which is changed according to an aggressive timetable.
3. Customer resource accounts would by default be established as sudo accounts. Sudo met their needs for authenticating as a resource account and allowed me to address the cost-prohibitive nature of deploying Powerbroker to 5000+ clients. if the account would be used to access what we considered to be mission critical or top secret information, then the account would be established as a powerbroker account.
4. If requested by the customer, we would provide powerbroker access instead of sudo for a resource account. This type of request was most often solicited by customers who were having resource accounts enabled for accounts which could impact the critical path progress of the company’s key projects – their need for accountability superceded their need for mass-access to the account across the environment.
5. We would not longer create suid programs for our customers. All scripts and programs would be ‘wrapped” around either sudo or powerbroker, again depending on the scale of need, and the importance of the information the script could affect.
6. All of the configuration files would be kept under RCS controlled and would only be modified by authorized members of the support team. The creation of entries for resource accounts could be performed by any member of the department, but the inclusion of new super-user accounts required approval by a member of the infrastructure and security team.

Productizing each of the solutions was easy to do at this point:

- ✓ The appropriate environmental change orders were filed with my customers who signed off on the proposed implementation.
- ✓ The above policy was then incorporated into the department’s documentation about resource accounts and the future use of powerbroker and sudo in the environment.
- ✓ Powerbroker’s master server services were landed on a permanent, hardened server.
- ✓ The owners of eligible accounts were contacted and offered opportunity to migrate their accounts from powerbroker to sudo.
- ✓ The configuration files for each product were finalized to reflect the current policy assumptions of the organization.
- ✓ The new binaries for each solution were deployed to the environment.
- ✓ All the appropriate users were notified of the new binaries.

"After"

After all of the procedural and technical details were implemented, the risk associated with the previous authentication and authorization strategies was significantly reduced:

- ✓ From a procedural standpoint, the establishment of departmental guidelines for the use of shared resource accounts, super user accounts, and suid programs enhanced our security posture because these policies were now well documented and provided all members of our support organization with a coherent process to be followed uniformly.
- ✓ The majority of the shared passwords were removed from the environment in lieu of sudo or powerbroker authentication strategies. This allowed the environment to evolve from discretionary access control to a mixture of role-based and rule set based access control[19]. Where authentication was once based upon a collective secret (the account password), authorization was now based upon either inclusion in a unix group (which could only be obtained through a managerial approval loop and inclusion in a unix group) or through specific rules approved and created by a subset of system administrators.
- ✓ Casual feedback from my peers indicated that more were willing to utilize sudo for their root executions because the extended ticket allowed them to operate without being hindered by the need to constantly provide a password.[20]
- ✓ Most noticeable was the reduction in the number of passwords which were cracked by my routine password file examinations with the password cracking tool, john the ripper.[21] Prior to this project, I would on average crack approximately 30-50% of the shared passwords. Now that almost all of them have been removed, they are beyond the reach of password crackers, and help in the overall password audit metrics reported back to the company's management.

At this point, I am satisfied with the initial results of this deployment. While it did not completely resolve all of the authentication and authorization risks identified at the outset, the implementation reduced the risks sufficiently to justify the investment in resources. Arguably there still exist threats that this project did not completely mitigate, but as the SANS track 1 indicates, "You can't protect against all threats"[22]. As such, I believe we have addressed most of the high risk areas

[19] SANS Security Essentials, Chapter 9, p 389
[20] As a side note, at this point we also began to use root passwords which were not as intuitive in an effort to coerce the administrators to use sudo over su.
[21] John the Ripper is a fantastic tool for performing an audit of your password file. I could write several pages singing its praise. Instead I would encourage those unfamiliar with it to examine it further at: http://www/openwall.com
[22] SANS Track 1 lecture notes. Chapter 7, page 303

and reduced each by at least one category (Very High, High, Moderate, Low). The deployed technologies also will provide for follow-on projects (which I will detail further below) that can further enhance my defense strategy.

The below table summarizes how these actions have impacted the Risk = Threat * Vulnerability equations that were introduced in the "Before" section. For most of these issues, the deployment was able to reduce the risk and/or vulnerability. This was a result of the changes these two applications introduced not only to the infrastructure (removing shared passwords, establishing logging, etc), but also to the procedural aspects of the project (the creation of definitive policy).

Table 5: Relative Risk "After"

Issue	Threat x	Vulnerability	= Risk	Orig Risk
A lack of a coherent policy	2	1	1	2
Suid programs	3	1	3	6
Insecure password sharing	3	1	3	6
Weak passwords	3	1	3	9
Password aging	3	1	3	6
Bypassing authentication	2	1	2	4
No accountability	2	1	2	4
No logging	2	1	2	4
No Separation of Powers	2	1	2	2
Session Hijacking	1	1	1	2

Next Steps

While implementing this project, several "next steps" surfaced that would further augment the project and the security posture of the environment:

1. Investigate the ability to remove the suid bit from OS binaries and replace it with a wrapper to sudo and a NOPASSWD option[23]. This wrapper would allow for notification and usage metrics as well as usage during strange periods of time.
2. Refine the log analysis scripts to provide patterns of super-user usage in the environment. Understanding these patters would improve our ability to delegate authority as well as detect anomalous behavior.
3. Modify the log analysis scripts to highlight super-user usage which violates policy – where possible send alerts to the user (and their manager) informing them of the violation and the proper methods for authentication.
4. Implement idled processes to monitor and remove root and other critical account shells which remain past an agreed upon timespan.

[23] I did not mention this option until now because I would not advocate its use except in clearly understood, very limited, circumstances. With the NOPASSWD option set in sudo, the user is not prompted for their password.

5. Modify the commands called in sudo to execute a checksum verification of the executables called – this would allow for greater assurance that the users are calling the programs they are authorized to execute.

Conclusion:

This project was initiated before taking Track 1. While I cannot claim that the course was the inspiration for its implementation, Track 1 did enhance the project remarkably. Before SANS, this was merely an identified security gap in the environment which needed to be addressed. Now it is a component to a defense in depth strategy and its implementation not only addressed the immediate identified risks but also made me consider this project in a larger scale that included inter-dependencies with other security projects focusing on detection and incident response. Before SANS, its deployment was based upon my unconsciously competent security knowledge; using the threat model I could now provide a justification that management could understand and buy into. The risk reduction demonstrated the ROI of the project as much as the reduction in password cracking metrics. My initial documentation covered the basic usage models and guidelines for what tool to use when; the information covered in the Basic Security Policy section of the course refined the content and what was included in this documentation.

Glossary[24]:

3DES	Triple DES. The application of the DES encryption algorithm 3 times in a manner to further increase the complexity of the final has.
Access control, Discretionary	User managed access
Access control, Rule Set Based	Access based upon a given set of rules
Access control Role Based	Access based upon a definable role
AFS	Andrew File System. AFS is a distributed filesystem that enables co-operating hosts (clients and servers) to efficiently share filesystem resources across both local area and wide area networks[25]
Authentication	The process of confirming the correctness of the claimed identity
Authorization	Empowering a user or process to do something
Availability	The ability of a process or system to provide an intended service to those authorized to access it.
CIDR	Classless Inter-Domain Routing. CIDR is aa addressing scheme for the Internet which allows for more efficient allocation of IP addresses[26]
Confidentiality	The need to ensure that information is disclosed only to those authorized to do so.
Cracked	The process by which a password is revealed through a variety of tactics and techniques
DHCP	The Dynamic Host Configuration Protocol (DHCP) is an Internet protocol for automating the configuration of computers that use TCP/IP. DHCP can be used to automatically assign IP addresses, to deliver TCP/IP stack configuration parameters such as the subnet mask and default router, and to provide other configuration information such as the addresses for printer, time and news servers[27].

[24] Many of these definition are either copied or paraphrased from the SANS Security Essentials glossary.
[25] http://www.angelfire.com/hi/plutonic/afs-faq.html
[26] http://public.pacbell.net/dedicated/cidr.html
[27] http://www.dhcp.org

Integrity	The assurance that a process or data has not been altered except by those authorized to do so.
Kerberos	Kerberos is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography[28]
LDAP	Lightweight Directory something Protocol. This is a protocol used to authenticate a user.
Need to know	Demonstration of a justifiable requirement to be authorized to access specified information or services
NFS	Network Filesystem. A distributed filesystem protocol
NIS	Network Information Services
One time passwords	An authentication scheme in which a password is only valid once.
PAM	Pluggable Authentication Model.
PKI	Public Key Infrastructure. The ability for users or services to exchange information through encrypted channels established using public and private keys.
Least privilege	Empower a user or service only to the degree necessary to perform some authorized action.
Risk	Based on threat and vulnerability, it is the likelihood of a successful attack.
Risk Assessment	The process by which risks are identified and the impact determined.
SecurID	A product which produces a one time password by coupling a pin with a sequence of numbers which is periodically altered. The sequence can be predicted by anyone aware of the algorithm used for its generation and the number the system was uniquely "primed" with.
Separation of powers	Division of power among multiple users. Just like the US government ☺
Session hijacking	The process by which an unauthorized user obtains control of system resources
SSL	Secure Socket Layer. An encryption protocol based upon public key strategies to transmit data over the network.
Threat	The potential for violation of security through vulnerability.
Vulnerability	A weakness in a system design, implementation, or operation that could result the exploitation of the system.

[28] http://web.mit.edu/kerberos/

References:

Miller, Todd. Sudo Website. 2003. http://www.courtesan.com/sudo

Powerbroker Website. http://www.symark.com/powerbroker.htm

SANS Security Essentials. Chapters 7-9. 2003

Komarnitsky, Alek, "Sudo at a large aerospace company". March 2003 http://www.komar.org/pres/sudo/index.html

D'Agnolo, Robert, "Implementing SUDO to replace SU: A Case Study Also Involving NIS and RDIST", SANS paper for GSEC certification, SANS 2003

Graham, Doug. "It's All About Authentication", SANS paper for GSEC Certification, March 15, 2003.

MIT. "Kerberos: The Network Authentication Protocol". Version 1.6. Aug 11 2003. http://web.mit.edu/kerberos/

Blackburn, Paul. "AFS FAQ". Version 1.37. June 10, 1998. http://www.angelfire.com/hi/plutonic/afs-faq.html

Packard Bell Services. "CIDR Info". 1999 http://public.pacbell.net/dedicated/cidr.html

Droms, Ralph. "Resources for DHCP". March 14, 2003 http://www.dhcp.org

"Common Vulnerabilities and Exposures". 2003 http://www.cve.mitre.org