The Problem of PORCMOLSULB :

Can Root be Controlled in Engineering Environments?

by Howard Owen, EGBOK Consultants, hbo@egbok.com

Introduction

The other day I got a call from a user who was having trouble checking in her latest changes to CVS. Since she was using a Linux box that was not supported by our group, I could have refused to look at the problem. But I'm pathologically interested in making sure our users get the most out of the computing environment. Besides, it could have been an issue with the CVS server, whose health as a system I am responsible for. So I strolled over to her cube and had a look. Our CVS check in process has hooks that cause email to be sent. I soon determined that the problem had to do with the fact that the sendmail binary on her system was owned by her. [1] Her boss, a senior engineer, had installed Linux for her when she started. He didn't ask the systems group to do the work because we would have set the root password to something he didn't know, and given the user sudo instead. (More about sudo shortly.) I walked over to his cube and asked him if he knew what was up with sendmail being owned by the user. "Sure," he said, "I did a 'chown -R <user> /usr" so she wouldn't have permission problems." I'm slightly ashamed to say I laughed out loud before telling him "Well, you are going to have to reinstall Linux." He got annoyed and asked "Why can't you just do 'chown -R root /usr'?" I told him why [2] and handed him the Linux CDs. They were back on my desk within twenty minutes, so I knew he had decided to implement his "solution" rather than reinstalling the system. This would solve her email problem, but other problems would surely be created. I told the user that I wouldn't touch her system until Linux was reinstalled.

I knew that the senior engineer in this case was no dummy, despite the incredibly bone-headed mistake he had made. He was in fact a very bright guy, engaging and witty in conversation and trusted with a critical role in designing the software our small startup [3] was betting everything on. What could account for the extreme wrong headedness he displayed? Why did he resist the reasonable restrictions we asked our users to accept in order to receive support on their personal workstations? How could I reconcile my certain knowledge that this was an extremely sharp and competent senior engineer with the apparently abysmal lack of wit his actions showed?

PORCMOLSULB

The above problem is an example of PORCMOLSULB: Proliferation Of Randomly Configured, More-Or-Less Screwed-Up Linux Boxes. It's been showing up more and more in environments I work in. This is partly due to the increasing popularity of Linux, but the main cause is the desire on the part of software engineers to control root on their personal workstations. This desire conflicts with the systems administrator's imperative to maintain systems in a supportable condition and to prevent anonymous damage to other systems on the same network from inexperienced root enabled users. The conflict is based not only on differing goals, but on real differences in the competencies and enthusiasms of the two groups. PORCMOLSULB adds an interesting new twist to the struggle that tends to shift the balance of power toward the users in the ongoing battle. In this paper I will describe the battle in a little more detail, then ask and answer the question "can root access on engineering workstations be controlled in the face of PORCMOLSULB?"

[1] On most Unix systems, the sendmail binary has the "setuid" bit set. This means that when it is run, it takes on the system identity of the user that owns the binary. Usually this is the root user, or some other system account that has permission to write to the mail spool, where temporary mail files are kept. Since the user didn't have this permission, and since the sendmail binary was owned by her, sendmail couldn't write to the spool and mail was broken.

[2] In any Unix-like system, the files stored in /usr will be owned by a variety of users. There is usually a reason why a particular file would be owned by a particular user. The example of sendmail being owned by root given above is one case-in-point. Once the ownership is changed through a global chown as in this case, it's very hard to set things back the way they were. It's easier just to reinstall the system.

[3] I work for a small startup company in Silicon Valley. The company wants to keep their name out of this article. Since I am telling potentially embarrassing stories about our engineers, I wholly agree with this position. The arrangement also allows me to be more frank about certain things, as long as I remain circumspect about others, such as names.

The Conflict Over Root Access

I've worked as a system administrator for eighteen years, in academia, for government contractors and in private industry. In each of those environments I have found a peculiar local version of low-intensity warfare between the computer users and sysadmins. I hasten to add that this conflict was rarely the only characteristic of relations between the two groups, or even the defining one. Nonetheless, the conflict was always present in some form. The most common form I have seen this conflict take is the struggle over root access. There are compelling business, psychological and technical reasons why this should be so.

The Role of Business Imperatives

From the perspective of the business employing them, systems administrators and technical computer users such as software engineers come to work for the same reasons. That is, to make widgets, grommets, yo-yos or whatever else the enterprise is producing. However looking a little deeper reveals differences in the business roles played by the two types of employees. Generally speaking, businesses hire systems staff to ensure that their computing environments are maintained in a state fit for maximizing the productivity of the enterprise. Software engineers generally are employed to design and write products for sale. It is their productivity that the systems staff must maximize. [4] This difference in business imperatives affects a lot of the interaction between the two groups. Specifically, it shows up when a software engineer demands root access to get her job done. Granting the access may in fact help the engineer to be more productive, at least until she shoots herself in the foot with her rootly power. The sysadmin is bound to see a threat to the stability and security of the systems under his care, and to discount the possibility that any benefit might accrue from granting the access that couldn't also be accomplished with a less sweeping grant of privilege. Before I examine that in more depth, I'll tackle the most difficult to characterize cause of conflict between system administrators and their technical users: the personalties of the people themselves.

The Role of Personalities

I've always thought that the conflict over root access was particularly strange in the context of Unix software startups in Silicon Valley. It seems to me that Unix systems administrators and Unix software engineers have a lot in common. However the difference in business roles described above, plus differing enthusiasms and capacities, tends to lead bright people with an interest in computer technology down different paths.

Apologia

Since I'm a system administrator, I can't avoid telling this part of the story from that perspective. I've tried hard to understand my users, and I've gotten pretty good at it over the years, but the coloration my own place in the scheme of things will lend to my discussion of the personalities involved in this conflict is unavoidable. With that warning issued, I hope you will forgive the personal nature of the discussion that follows.

My Choices

Why am I drawn to system administration? Why not be a software engineer, for instance? I do a fair amount of programming in my work. I can code some Perl for several hours, enjoying all the things you must do to program effectively, such as holding several dozen details in your mind at once. Best of all, I love integrating all those details into a finished solution that actually *does* something. However, I don't like to wait too long to get to that point. I'm impatient. I also get burned out quickly doing that sort of thing. Finally, I get bored really, really easily. Fortunately, as a sysadmin I am compelled to do a lot of other things. I have to deal with other human beings, frequently under difficult circumstances. I work with computer hardware a lot, racking up systems or diving under desks to replace bad components. And best of all, I get to work with computer systems. Unix, Linux, Windows, Palm, it doesn't matter. I love systems. I can make them stand on their heads or dance the two-step. I love the feeling of control and accomplishment that going to the exact center of a difficult problem in complex systems gives me.

[4] This is a sweeping generalization. There are plenty of systems engineers directly contributing to product, just as there are many software engineers working on the productivity of others. However I'll stand by the generalization for the purposes of this discussion, since my experience tells me it's true more often than not.

My Users' Choices

How is all that different from what a typical software engineer does? This is hard question for me to answer, because I have to try to put myself in the place of an engineer, and I tend to just assume that she thinks exactly the way I do. However there are some clues in the experiences I've had with such engineers that have helped me make the leap of imagination. First, I've noticed that these folks seem to have powers of concentration that are rather absurd, by my standards. Whereas I need to take a break after a couple of hours of coding, these guys stay glued to their screens and keyboards throughout their fourteen hour days. Second, I've noticed that their technical knowledge tends to be less broad than mine, but deeper. Both of these observations start to add up to a (perhaps) obvious conclusion: *software engineers are specialists.* Another conclusion I've drawn has taken a lot longer to arrive at, because it cuts so directly against my own stance toward technology. *Software engineers are generally not enthusiastic about computer systems.* Instead, they are enthusiastic about *software!* They view systems as a vehicle for software, a means to an end. I view systems as ends in themselves. Once this idea struck me, I marveled at how long it took me to see it. It seems that both systems administrators and software engineers are constitutionally suited for the differing roles they are asked to play in the enterprises that employ them.

Now that we've introduced the players, let's set the scene: Unix in all its common permutations, including Linux.

The Role of Technology

Unix operating systems generally provide a rather primitive model for distributing privilege to system users. The power to control all system processes and resources is granted to the single all-powerful user, root. Other users may be granted varying levels of access, depending mainly on which Unix group they belong to and on how group access permissions are set on various objects in the system. However root (or any user with UID 0) is the only user that can arbitrarily change access permissions. As a result, when non-root users encounter a restriction in access permissions, they must call upon the power of the root user to rearrange things so that they may continue their work.

C2

There are exceptions to this monolithic permissions model among various proprietary and free Unix implementations. Many OS vendors, including most Unix vendors, have applied for and received DOD Orange Book C2 certification for one or more of their products. (For an exhaustive list see http://www.radium.ncsc.mil/tpep/library/fers/tcsec_fers.html.) However these vendors generally do not ship their systems with C2 security enabled. Even Microsoft, whose Windows NT code base implements many of the facilities that C2 requires, such as Access Control Lists, doesn't do that. Since Microsoft, at least, has had to submit a version of NT with network access disabled in order to get certification, that's not entirely surprising. And though I'm not an expert on the topic, I suspect that the reason even those vendors who may be able to run C2 while on a network don't ship with it enabled is because C2 access controls are fairly burdensome to users and administrators alike. Regardless of the real reason, the fact remains that the Unix systems found in most commercial environments, from vendors like Sun, HP, IBM, as well as Linux and xBSDs, come configured by default with an antiquated permissions model.

Working Within the Model

Even given the monolithic Unix permissions model, it is possible to give users most of what they want without unleashing the full power of root. Issues that concern shared access to files can be dealt with by judiciously adjusting group membership and permissions. If a user needs to open low-numbered TCP/IP ports, for example, it's possible to setuid root just a particular binary, though that carries with it all sorts of other security implications. There are many strategies that help users to "work within the system." Each of these has in common one fatal flaw, however. If the model needs to be adjusted because of an unforeseen condition, root (a.k.a. the sysadmin) needs to get involved to make the adjustment. In a rapidly changing environment like a software startup, this has several impacts on the user. First, it slows her down. A overworked and harassed sysadmin has to be located by an at least equally overworked and harassed engineer to make the change. According to Murphy, this will always happen at 3:00 AM on the morning before a critical demo. I *really, really hate* to have my cell phone ring at that hour! Even if that apocalyptic scenario doesn't get played out, resentment may be fostered on both poles of the struggle. The user may start to see the sysadmin as a power-mad tightwad, jealously guarding root access for his own nefarious purpose. The sysadmin may feel put out by the fact that the user isn't willing to learn enough about Unix to get around her problem. He may also be blind to any benefit that might accrue to the user and the enterprise from allowing the access. There's a bit of irony here. Systems administrators often complain (with justification) that what they do is never visible until something breaks. This is a consequence of the natural outcome of great sysadmin: quietly working systems. In a similar way, the sysadmin is unlikely to see any benefit from giving a user root, because those benefits short-circuit trouble calls to the sysadmin!

Before moving on, I'd like to note that neither of the characterizations presented above is fair, and they rarely play out in such an extreme form in the real world. But their flavor is correct, at least to judge by the places I've been.

Tools That Try to Help

Because working within the system is so troublesome, the ever-inventive Unix community has produced many tools [5] that try to add finer-grained control to the monolithic Unix permissions model. One of the most popular is sudo. This tool is familiar to many sysadmins. It allows users to invoke specific commands with root privilege. It uses the user's own password to authenticate access, thus protecting the root password. It also logs each command invocation with the name of the user, thus providing an audit trail of root access. Typically, tools like sudo are deployed to meet a specific user need for root access, such as to enable them to change permissions on directories. However, security issues can arise from the use of such commands that allow the user to evade the restrictions sudo provides, either innocently or deliberately. For example, the chmod command that can make an inaccessible directory available could also allow the user to set the setuid bit on a binary. If that binary is /bin/bash, or any other shell, and it's owned by root, as is usual, running the shell after a 'chmod 4755' would result in an unaudited root shell. One way around this would be to wrap a script around the chmod command that disallowed setuid mapping. But then you have to worry about the security of shell scripts running as root. In fact, in a relatively open environment like a software startup, there is no sure way to protect yourself from malicious misuse of privilege in all cases, even without sudo. You end up having to fall back on trust, treating abuse of that trust as a personnel problem.

The problem gets even worse as more and more commands are added to the suite of those offered to sudoers. Each new command brings its own particular set of security holes. The problem, once again, is that Unix assumes a monolithic permissions model that tools like sudo can only work around, not cure. And if you allow shells to be run with sudo, you might as well just give the user the root password. This is because sudo only logs the initial invocation of any command. If that command is a shell, such as /bin/bash, subsequent commands issued from that shell are not logged.

The weakness in the Unix permissions model shows up again as problems with programs like sudo that have nothing to do with the quality of the code, and everything to do with the fact that hacks like sudo are necessary in the first place.

For example, sudo has difficulty with IO redirection:

```
hbo@egbok > ls -l /tmp/foo
-r--r--r--   1 root     other       1464 Mar 25 13:10 /tmp/foo
hbo@egbok > sudo ls >>/tmp/foo
bash: /tmp/foo: Permission denied
hbo@egbok > sudo ls | sudo cat >>/tmp/foo
bash: /tmp/foo: Permission denied
```

This problem occurs because I/O redirection is implemented by the shell before the command (sudo) is executed. The monolithic Unix permissions model leads the shell to assume that the identity that does the I/O redirection is the same as the one that will result from the execution of the command. This is false in the case of sudo, which violates that permissions model. The following trick gets around the problem:

```
hbo@egbok > sudo ls | sudo tee -a /tmp/foo >/dev/null
```

But it's not very intuitive. This also works:

[5] A comprehensive list of such tools is maintained at http://www.courtesan.com/sudo/other.html

```
hbo@egbok > sudo sh -c "ls >>/tmp/foo"
```

But as previously noted, if you allow shell access with sudo, you might as well give out the root password.

Globbing is broken too:

```
hbo@egbok > mkdir fff
hbo@egbok > chmod 700 fff
hbo@egbok > touch fff/foo
hbo@egbok > sudo chown root fff
Password:
hbo@egbok > cd fff
bash: cd: fff: Permission denied
hbo@egbok > sudo cd fff
sudo: cd: command not found # cd is a bash builtin!
hbo@egbok > sudo rm fff/*
rm: cannot remove `fff/*': No such file or directory
```

The 'globbing' expansion requested by the use of the asterisk fails because, once again, the shell tries to do it before executing the sudo command. We also see in this example the problem of trying to 'cd' into a protected directory. Since 'cd' is a bash builtin, sudo doesn't know what to do with it and you are out of luck.

Of course, you could put code to solve either problem in a script. But if you let your users run, for instance, Perl with sudo, what's to stop them from writing something like this?

```
#!/usr/bin/perl
exec "/bin/bash";
```

Once again, there goes your audit trail! In fact, if your users have successfully agitated for sudo access to more than a handful of commands you will almost certainly face an impossible number of holes in your security policy.

PORCMOLSULB, Again

So far, we've seen two groups of professionals, apparently similar on the surface, engaged in a struggle for control of root access on personal workstations. Each group is trying to carry out the goals that their respective business imperatives demand. The software engineer wants root so that she can get around restrictions in the Unix system in order to get her work done. The system administrator is trying to ensure that the user's system stays functioning. What are some possible outcomes of this struggle? Complete victory by either side is unlikely. To borrow a concept from chemistry, a more plausible outcome is that some sort of "dynamic equilibrium" will be reached as managers in support and engineering struggle to balance competing business interests. When the struggle concerns root access on servers, the business imperatives lean more toward the sysadmin's view of things, because the technical problems of sharing root on a server are less tractable, and the consequences of a root compromise more obvious. On engineer's personal workstations, however, the business case for allowing unfettered root is more compelling, because the workstation is a primary tool enabling the engineer's productivity. If the balance of power shifts toward the sysadmin, we start to see the phenomenon of PORCMOLSULB showing up. This occurs when support departments can't keep up with the demands of their user base for development "play-pens," or when they put restrictions on those play-pens beyond what the users are willing to accept. It turns out that engineers are increasingly able to convince their managers that a completely uncontrolled Linux box would be a boost to their efforts in the rush to meet insane deadline pressure. [6] The sysadmin crew is probably feeling the pressure too, so they are in worse shape than normal to resist this trend. Indeed, they may not even become aware the box exists until it shows up in the critical path for some important milestone. If support departments know the box is being deployed, but lack the power to prevent it, they frequently disavow support, citing the extreme difficulty of maintaining a box with a random configuration. But in a software startup, they can still find themselves stuck with fixing the box under killer time pressure, with the business on the line and with no advance idea of how it was configured by its amateur sysadmin.

[6] Managers generally feel bad about asking their people to work 12+ hour days to meet unreasonable deadlines, no matter how they put a brave face on the matter. Giving their engineers the tools they need is therefore not only good sense from an organizational standpoint, it lets the managers hand out a perk or two.

What is to be Done?

That nightmare scenario didn't actually happen to me in the case I opened this paper with. But we *were* facing a killer deadline, and the mere possibility of it happening made me nervous. I had faced similar situations before, so I knew that arguing for the "right" way of doing things wouldn't lead me anywhere useful. I'd also recently had my epiphany regarding the surface similarity and deep difference between sysadmins and software engineers. Here's how this particular comedy *did* play out.

Do you sudo?

About 10 days after the senior developer got his engineer working again by doing a 'chown -R root /usr', she showed up in my cube asking for the Linux disks. I was mildly surprised that it had taken that long for a side effect of that solution to convince her that she needed to reinstall. But I tried not to act smug, and handed her the disks without asking why she wanted a reload of Linux. But I did ask her if she wouldn't rather that I do the install. I'd set her up so that her home directory on her workstation would automount underneath her when she went out to the network. I'd also arrange for it to be backed up regularly, and support it so that she could come to me if she had problems. She allowed as how that might be a good thing. So I delivered the punchline: "All you have to do is give up root and use sudo. It takes a little getting used to, but I'll help out." Well, she readily agreed to that too, and I was in a self-congratulatory mood when she came back in ten minutes saying her boss had nixed the idea. He said she had to have root instead of sudo.

I actually took a short time out before going over to his cube. My question to him was rather sharp, but nothing like it could have been. "Do you really think backups, the automounter and support are worth having the root password?" "Yes," he said. "*Why* fer ----ssake??" I politely asked. "Because you won't let us run shells with sudo!" I proceeded to tell the story of 27 eight-and-a-half by ten colored glossy audit trails.. he said "Stop right there! What good would an audit trail do you if someone did 'chown -R <user> /usr'?"

Well, he had a point. But I had an answer: "because the audit trail would tell me right away that I had to reload Linux, rather than some less drastic solution. And besides most problems aren't caused by thoroughly bone-headed moves like that one!" He laughed and said "OK. Give her sudo."

I felt pretty good after that. It could have turned out differently, but it didn't. Despite the sharpness of the exchange, I felt like I'd made a critical connection with this guy. [7] In addition, I had a toehold in his group with a supported Linux box that would *not* be randomly configured, and would be less, not more screwed up. And his new engineer would be using sudo! I would work hard to make sure that she had as good an experience with it as I could manage.

In fact, over the next couple of days they came to me several times with things they couldn't do with sudo, and could I please just run the command with root? Each time it had nothing to do with sudo, and each time I cheerfully fixed it for them, or pointed them in the right direction. Soon, I had a couple of converts.

Beyond Sudo

Now it turns out that the senior developer, and all his colleagues, were resistant to using sudo because we restricted shells. This is an area where a sysadmin can argue unto blue-facedness about the lack of a need for a shell when you have more-or-less unrestricted sudo access. Indeed, since we had such unrestricted access, escaping from sudo and its audit trail was a trivial exercise. Given those facts, I decided to just accept that despite the technical arguments, sudo alone was not a workable solution for these senior engineers on their workstations. In half a day, I whipped up a pair of Perl scripts that used script(1) and a fifo to provide an audited root shell using sudo. [8] This gave them practically nothing they didn't have already with our open sudo policy, and preserved our audit trail. All the senior engineers accepted a support regime which included these scripts.

[7] I probably neglected to mention that I'm new on the job.

[8] The result of considerably more than half a day's effort is available at http://www.egbok.com/sudoscript.

Caveats and Conclusions

Danger [W|J]ill Sysadmin!

There are *big* problems with this solution. Having root on a workstation that mounts NFS shares is tantamount to giving the user root on the NFS server! Most NFS servers can and should be configured so that any access to an exported file system by UID 0 is mapped to a user with no privilege whatsoever. But that's not the whole answer. With root, a user can assume any UID in the passwd map. This means that on the NFS server, other users' files and system files not owned by root, are at the mercy of root *on the NFS client!* The approach I've described works best when the workstations are NFS servers, not clients. There is still an issue with other systems mounting shares from the workstation. If the NFS client implementation doesn't enable you to disallow setuid binaries, a root user on the workstation could place, for example, a setuid root bash binary on the exported file system, then execute that binary on the client and get root privs.

Philosophy 1001

This is not an exhaustive list of the security problems such a setup could raise. But remember that in my environment, the users had successfully agitated for more-or-less unrestricted sudo access. Since this was so, giving them an audited root shell didn't raise any more security concerns than were already present. So I'm not claiming that this would work everywhere. But, in my small shop, I can look each of my users in the eye every day if I choose. There is not a single unteachable idiot in the bunch. I also don't hand out my scripts to everyone. In short, I rely on the good faith of my users. I give them the tools they say they need, and I try to give them the benefit of the doubt on the question, despite my technical knowledge to the contrary. If they shoot themselves in an extremity with their privilege, I triage and fix the damage, with the benefit of a recent audit trail.

```
<tirade mode="self righteous" color="purple">

  I trust my users in this regard because of one argument in favor of
  Democracy: If you give people more choices, some will make bad ones.
  Many more will make good ones, yielding a net benefit.

</tirade>
```

And this principle may well be applicable beyond my environment. How it plays out in yours is up to you, and your users.

Finally, Documentation

Once I've fixed any problems caused by inexperienced root users blasting off their toes, I try to leverage the occasion to get them to read my documentation. Ah, yes. Documentation. Nobody likes to write it, and nobody likes to read it. I write lots of documentation and perversely perhaps, I enjoy doing it. What I find hard to take is the indifference most of my users show toward what I write. My epiphany regarding the differences between sysadmins and software engineers has provided me with an explanation for that conundrum as well. *What's relevant to me and to the systems under my care is not directly relevant to my user's concerns!* If I were to write the best-ever UML manual, *then* they might notice. But when a pretty bright engineer has made some embarrassing error that has clearly resulted in a hit on her productivity, or worse, that of her colleagues, then the docs I write may seem more relevant. You have to be tactful and swift in exploiting these opportunities for education, however. Tactful because these folks are proud, and their pride has just been wounded. Swift, because they'll have their heads completely stuffed full of Java before long, with little room for anything else.