Giving Customers a Tool to Protect Themselves

Shabbir J. Safdar
shabby@mentor.cc.purdue.edu

Purdue University Computing Center
West Lafayette, Indiana 47907

ABSTRACT

For those who administer security, keeping an eye on passwords and system software is only the first step. Security of customers' accounts is the next step. In a computing environment with a rapid turnover of customers, this is quite a challenge. One cannot simply hand every customer a UNIX† manual and assume all is well. At a site with a large population it is also difficult to simply scan every account and educate every customer when a problem is found. Customer account security should be a shared responsibility, with no undue burden placed upon the novice customer. However, there is a definite gap in the area of security for novices. Other topics have manuals or programs which take the customer by the hand in learning the basics, such as *learn*(1). What is needed is a utility that takes some of the repetitive work out of educating and teaching customers the basics of account security. A site with a significant number of customers in the habit of using such a utility would increase security on their accounts without requiring the intervention of an administrator. Such a utility should be extremely security-conscious, defaulting to the most conservative choices possible. It should be easy to use, look the same everywhere, and provide as little or as much information as is desired by the customer. Finally, it should have optional facilities to educate the customer if that is desired.

1. Introduction

In the rush to protect the security of the "super-user" account it is easy to treat security of individual customer accounts as a lower priority problem. However when a customer's account has been broken into, it does not matter whether the attacker gained access by compromising the "super-user" or through a security problem in the customer's account itself. Either way the result is the same: they have lost privacy and perhaps some or all their files. Thus far, the solution to this problem has come in the form of programs that require the system administrator to check various aspects of customer account security. Three examples of administrator utilities to improve customer account security are:

- *home.chk* from the COPS security package distributed by Dan Farmer (zen@death.corp.sun.com).
- *user.chk*, also from the COPS security package.
- *crack*, a password file cracker distributed by Alec D.E. Muffet (aem@aber.ac.uk).

While these three are excellent programs for the system administrator wishing to improve site security, they do little to educate the customer on the basics of account security. Any education resulting from the use of these tools is done by the system administrator after the results of the

† UNIX is a trademark of Bell Laboratories.

utilities have been examined. The other disadvantages of these tools are that:

- They must be run often to be of use on an active machine full of curious customers.
- They compromise the privacy of the customer's account to check file permissions. (albeit in the process of trying to ensure further privacy)
- They must be run as the "super-user" so that closed accounts can be checked.

An alternative solution to these problems is to give customers the tools they need to do a significant amount of account security checking on their own. The design of such a tool has the following requirements:

- It must assume the customer has little or no prior knowledge of account security.
- It should have facilities to educate the customer if she or he wishes to learn more about account security.
- It should default to be as security-conscious as possible.
- Since novices will be the primary clients of such a tool, it must possess the same look and feel across several different architectures.
- It must be easily configurable to different sites.

One implementation of such a design is the customer utility *chkacct*, short for check account, developed at the Purdue University Computer Center (PUCC). *chkacct* was written in Bourne shell using standard UNIX utilities. When invoked by a customer, *chkacct* examines several aspects of the account, offers a solution, and provides helpful informational files‡ explaining basics of account security on the particular topic in question.

2. Security From the Customer's Point Of View

Security, from a customer's point of view, is:

- System software security
- Backups
- Passwords
- Customer file security
- Data encryption

There is already a large pool of information on how to administer secure passwords, system security, secure encryption, and backups. These all assume that the system as a whole will benefit from a "top-down" approach, where the security of the system benefits the most from the security of the system software. However when all the bad passwords have been cracked, and the vendor-shipped world writable files have been caught, a security-conscious administrator is left with the overwhelming task of keeping the customers out of each others' hair. Left unchecked, security on individual customer accounts deteriorates. A world writable file or two later, and a customer has been trojan-horsed.

If there was little interaction between customer accounts, one might make a case for not acting on security problems in customer accounts. However this presumes that we neither care about our customers' data nor that our customers interact with each other. Neither of these statements is true at PUCC. Our customers often install software packages in their own accounts which they share with other customers. At last count there were 17,000 people with accounts on one of our machines. Because we supply "career accounts" to full-time students, it is likely that each of these people has more than one account, many of which trust each other.

So it seems that we are obligated to do customer account security. However that does not mean we have to do it in an unintelligent manner.

‡ provided by Phil Moyer (prm@ecn.purdue.edu) of Purdue's Engineering Computing Network

3. Methods of Breaking Into Customer Accounts

There are four popular methods for breaking into customer accounts seen at our site. They are:

- Revealing one's password to another customer
- Insecure file permissions (world writable files, etc)
- Trusting another account (which trusts two more, ad nauseum)
- Trojan horse programs

The first problem, password revelation, is by far the worst, most rampant problem at PUCC. Because we do not place limits on monthly cpu and connect time usage, there is no reason for anyone who trusts "a friend" not to give away their password. Occasionally someone also is not careful about who they type their password in front of, and they give it away without realizing it. The problem of trojan horses is either so subtle we are not aware of it, or just infrequent at our site. Either way, we do not come across many of them. The problem of trusting other accounts (often via *.rhosts* file) has been dealt with only minimally so far. Although we have policies against account sharing, they are not enforced enough, nor are they broken brazenly enough to elicit an administrative crack down. Occasionally someone will be so blatant about it that they will get called in and have the policy explained to them in greater detail.

Of the four methods mentioned above, the one that held the greatest promise for improvement was insecure file permissions. We felt that this problem was where we should concentrate our efforts.

4. Popular Security Methods & Their Drawbacks

One response to such a problem would be to run a security tool which checked our customers' accounts for us. COPS might be such a tool but even it has limits of what it knows. For example, it will not discover a shell script owned by someone within a *bin* directory. Let us assume that some time in the future someone releases to the public domain his ultimate security tool: MOPS (Mighty Omniscient Protection System) MOPS is so advanced that it will peruse all of the customers' accounts, notifying the administrator of every possible security problem. It will parse apart every *.rhosts* file checking for untrusted accounts and examining every file and directory. Even with such a security tool, there remain several problems.

4.1. Administrator Intervention

In the end, MOPS still requires the administrator to stay "in the loop" to verify problems, educate customers, and take care of problems which they do not take care of themselves. This often requires a certain amount of dialogue with the customer. Since ours is not the sort of site where we can just haul off and play with customers' accounts and apologize later, we would be doomed to process the output of a MOPS package themselves. This administrator intervention is both time-consuming and a poor use of human resources.

4.2. Do Not Run MOPS Too Often

If run often MOPS becomes a high-profile piece of software which can be both evaded and used as a pointer towards insecure accounts. To do the latter, one would just to examine the mail logs at the known time at which MOPS was run every night, or hunt through them for letters of a similar size to the form letter that MOPS sends. This even works if the staff sends these letters out by hand, as we do.

4.3. But Do Not Run MOPS Too Infrequently

If MOPS is not run often enough, then it is of little use to the customer, since by the time they are notified of a problem, other attentive and malicious customers may have already exploited the security problems in question. So it seems that there is no optimal time to schedule a MOPS run, since the optimal time to run MOPS over a customer's account is simply *at any time the customer wants to check the status of their account.* If a customer were able to check the security of their account at their own convenience, then this would be the first step to

helping them make the connection between the actions they perform and the security problems which can be the side effects of these actions. However the MOPS approach to security does nothing to encourage customer responsibility. In fact, when the customer comes to depend upon the administrator to take care of security, educating the customer on account security becomes more difficult.

4.4. MOPS Is Intrusive

Security methods which examine customer accounts are intrusive. MOPS is, by definition, intrusive. However privacy is a perceived condition. Many customers are unaware that anyone with "super-user" access can read any file on a standard UNIX system. By sending customers letters to asking them to attend to security problems in their accounts, one reminds them that someone or something is regularly looking through their files. This paper takes the side that some sacrifice of privacy is a necessary trade-off for security. However, if one can depend on the customer to check most of their personal files for security problems, then one can minimize the number of personal files that MOPS checks, as well as the amount of time the MOPS administrator spends administering security.

4.5. Summary: MOPS Is Not A Sufficient Tool For Customer Security

Security of the operating system is the priority of a package such as MOPS. However simple security of the system is only the first step. Customer account security is the next. It is not sufficient or wise to attempt to administer customer account security with warnings followed by dialogue. Not only is this a poor use of an administrator's time, but it is also a solution which does little to improve the problem. Part of the problem is with the approach taken towards customer account security. Customer account security should be a shared responsibility, with no undue burden being placed upon the novice customer. The following analogy may help clarify this approach.

5. The "Desk" Analogy

A multitude of people have drawn parallels between physical security situations and electronic ones. Bear with me while I make one more: the office desk. We all practice intelligent security in regards to our office desks. If we are concerned about the contents of our desk, we lock either it or the office door. We do not let anyone in our desk or office that we do not trust. When something comes across our desk that is important enough, we photocopy it and store it in a secure place, such as a separate drawer or a safe. And we always keep confidential or personal information in a locked filing cabinet or box. Furthermore, we assume that only we and perhaps the building supervisor have keys to our desk or office, and we trust that they will not give the key to anyone else. Lastly, our desks are probably in somewhat secure areas, where someone from the street cannot simply walk in and rifle through them. There is probably something to stand between them: a night guard, a locked office door, or even something as meager as a push-button lock on the office door.

We feel fairly secure working at our desks, but imagine if our desks were computer accounts on a MOPS monitored system. If we are at a site on the Internet without a strong network firewall, anyone can walk right up to our desks and play with the lock. Assuming our administrator takes care of our backups, we are trusting someone else to come and look through our desk for all the documents that have changed, and then photocopy them every night. If we are novices, we do not have the ability to spot when we have left either our desk or our office door unlocked. We just assume that its safe, and if it is not, that the building guard will come by later on and look through our office and desk. If the guard finds something that is insecure, we will be left a private note, giving instructions on how to secure the office or desk drawer in the hopes that we will see it and follow them.

There must be a better way. When an administrator uses a system such as MOPS for customer account security, the number of customers is proportional to the amount of time required by the administrator to perform customer security. Few large facilities can afford to devote a large

amount of staff to such an endeavor.

6. Deriving Customer Responsibility From The "Desk" Analogy

This analogy seems a bit silly when we apply it to the physical world, but it does demonstrate how inefficient and labor-intensive this approach to security is. It is important to note that exactly what makes this analogy ridiculous is that we all know how to keep our desks secure. However not as many know how to secure our own UNIX accounts. This is true at our site, and I imagine it is true at many other sites as well.

It does not seem unreasonable to expect customers to take some steps to maintain security on their accounts. What does seem unreasonable (or at least unlikely) is to expect every first semester FORTRAN student to learn enough about UNIX to be able to not only notice every potential problem, but also to know enough to fix them. For many of our students, they have but one class on a UNIX system to take during their entire enrollment period. For whatever reason, they dislike computers. So it seems unrealistic to assume that they will expend any effort or spend any time on our systems that is not absolutely necessary to passing their class.

As is demonstrated by the analogy above, it is not an easy task to have one administrator perform MOPS-style security for a site. Even if it were possible, the sections above show how MOPS-style security, when applied to a customer population, distances the customers from their security problems, making them less aware of security issues. What is needed is a "customer-oriented" approach to security targeted towards novices.

7. A "Customer-Oriented" Approach to Security

A customer-oriented tool for account security should scale itself to accommodate both novice and experts. When it finds a security problem, it should provide a default action which is security-conscious. It must be portable and look very similar across different platforms. *Chkacct* was written with these requirements in mind. Once invoked by the customer, *chkacct* examines an account in three phases. The first phase checks the permissions of all "dot" files (files such as *.login* , *.rhosts*, *.profile* etc.) Working under the assumption that all "dot" files contain the most sensitive information, *chkacct* warns the customer about "dot" files which should not be either readable or writable. *Chkacct* also flags any remaining "dot" files residing in the customer's home directory, but owned by someone other than the customer running *chkacct.* The second phase examines all files owned by the user running *chkacct* (including directories) for writability, setuid (set user id), or setgid (set group id) permissions. The third phase of *chkacct* is a *perl* script which attempts to parse apart the customer's *.rhosts* file, if it exists. If it exists and is found to be unsafe, *chkacct* offers to move it to another name so it will not allow any password-less logins. Lastly, *chkacct* offers to display an article about account security. The article is written for novices.

Chkacct satisfies the five requirements set out in the beginning of this paper. First, because *chkacct* is targeted towards novices, it requires no previous knowledge of security. Secondly, *chkacct* contains facilities to educate the customer. If one wishes to learn more about the particular topic in question, that facility is available. Thirdly, it is targeted at novices since the default actions are security-conscious. Someone with no knowledge of UNIX and a trust of the *chkacct* program would still find their account in better shape than before they invoked it. Fourth, *chkacct* is written with standard UNIX utilities, so it looks similar across platforms. Finally, it is configurable to site specifics, in that one can specify things such as "guru" and "consultant" names, whether or not a site uses group permissions, etc.

8. Examples of *chkacct*

What follows are some examples of *chkacct* responses to various security problems. They are included to give the reader a sense for the novice level that *chkacct* is targeted to. The initial screen of *chkacct* attempts to prepare the customer for the examination of their account. It also provides a good opportunity to pause and let the customer digest what they are doing. This first example is what one might see if their *.login* file was world writable. Initially the file name and

the problem is presented to the customer, along with enough diagnostic output to allow a more experienced UNIX customer to decide if this is a problem or not.

```
File '.login' is world or group writable.
The output of the command "ls -gld .login" is:
-rw-rw-rw-  1 shabby   pucc   1543 Jul 22 21:59 .login
```

Then a suggested fix is shown. In the case where the file in question is a "dot" file, a short explanation is also automatically displayed.

```
The suggested fix for this is to execute the command:
 /bin/chmod go-w /userb/shabby/.login;

Most accounts have special files called "dot" files.  These
files control the startup, environment, and execution of the
shell and some programs.  It is very important that these
files not be writable or owned by anyone but you!  If someone
else owns or can write those files, they can take control of
your account in a matter of minutes! Then they will be you,
which means they can do anything you can do: read, write or
modify files; send mail; talk to other users; print documents.
Make sure that permissions on these files are set to 644, or,
better yet, 600:

 .login      .logout     .cshrc     .bashrc     .kshrc   .exrc
 .xinitrc    .dbxinit    .profile   .sunview    .mwmrc   .twmrc
```

Now it is time for our customer to decide what to do. A menu is presented allowing for several choices. The default action would be to fix the problem, which is what our example customer happened to do.

```
Press a letter (a) to enter automatic mode (no more questions),
(f)ix problem, (h)elp me out with this menu, (i)gnore problem,
(m)ore info
Press RETURN/NEWLINE to fix the problem and go on>
Fixing problem...Done.
```

In this second example, a setuid file is found in the customer's account. Once again, the problem is presented to the customer with some diagnostic output:

```
Your file .super-secret-sh is user or group setuid.
The output of the command "ls -gld .super-secret-sh" is:
-rwsr-xr-x  1 shabby   pucc 169399 Jul 22 22:02 .super-secret-sh
```

A suggested fix is provided along with an explanation about the effects of such a fix:

```
The suggested fix for this is to execute the command:
 /bin/chmod ug-s .super-secret-sh;
which means that when someone else executes this file, they
will NOT gain your account permissions.
```

And once again, our example customer decides to go with the default:

```
Press a letter (a) to enter automatic mode (no more questions),
(f)ix problem, (h)elp me out with this menu, (i)gnore problem,
(m)ore info
Press RETURN/NEWLINE to fix the problem and go on>
Fixing problem...Done.
```

The last example shows the output of the third step of *chkacct* which checks the *.rhosts* file. Once again, the problem is shown:

```
These users at drop-dead.cc.purdue.edu are allowed to login
to your account without a password: fred

Your .rhosts file is unsafe.
The output of the command "ls -gld .rhosts" is:
-rw-------  1 shabby   pucc   29 Jul 22 22:07 .rhosts
```

A suggested fix and its ramifications are also presented:

```
The suggested fix for this is to execute the command:
 /bin/mv -i /userb/shabby/.rhosts /userb/shabby/rhosts.4657;
which will prevent anyone from logging into your account
without a password.  After talking to a PUCC Consultant
(available in the basement of Math-Science or at 49-41787)
you can edit this file, rhosts.xxxxx and move it back to be
your effective rhosts file.
```

9. Statistics On *Chkacct*

9.1. How We Use *Chkacct*

At the Purdue Computing Center we run COPS nightly. When security staff is alerted to a problem in a customer account, for example by *home.chk*, we send the following form letter to the customer:

```
To: recipient()
Cc:
fcc:
Subject: Security on your PUCC Unix account
------


You have files in your account (recipient()) whose permissions
are insecure.  As a consequence, other users may be able to write
on them, modify them or even delete them.

The program chkacct(1) will scan your account's directories for
insecure files.  Each insecure file and the reason for its
insecurity will be explained to you, and you will be given the
opportunity to select an action that will make the file more
secure.  You can run chkacct(1) by typing the following command:

    $ /usr/local/bin/chkacct
```

chkacct(1) will scan your account's directories, select files with questionable security, rate the relative danger of their insecurity and offer you options for making them more secure. If you are unsure what option to select, just press RETURN and chkacct(1) will make the file as secure as possible. (chkacct(1) opts for maximum security by default.)

If you wish to read more about file permissions, there is an article that can be displayed (by typing the word "yes" when prompted) at the end of the chkacct(1) program.

It would also be prudent to add the following line to your .login or .profile:

```
umask 022
```

The effect of adding this line is that all files you create from here on will be writable only by you, but readable and executable by other users.

If the problem persists after a few days then we intervene, fixing the problem for the customer and then sending a letter explaining what changes were made. In most cases this is simply the changing of permissions on a file such as a *.login*, *.rhosts*, or a home directory.

9.2. Collection And Analysis Of Data

This practice began in December of 1991, when *chkacct* was first installed on our systems. The cutoff of data collection for this paper was July 1, 1992. Although *chkacct* was not written with a future statistical study in mind, we were extremely fortunate, since the collection period fell almost entirely within Purdue's academic year. Analysis showed that during the seven month collection period, one hundred and forty nine separate accounts were sent warning letters about file permissions. Out of these, one hundred and eleven required no further action. Thirty eight did indeed require intervention, and letters were sent informing them of the file mode changes to the account. This means that three out of four customer problems were resolved in one of the four following manners:

- The customer possessed enough knowledge to fix it themselves.
- The customer ran *chkacct* and it fixed the problem.
- The customer asked a consultant or another person to fix the problem.
- Someone broke into the account and fixed it to avoid arousing suspicion.

We considered all but the last of these to be sufficient resolutions to the problem. Here is the data broken down on a month by month basis:

Warning Letter and Intervention Data for Dec. '91 through Jun. '92				
Month	Number of Warnings	Number of Interventions	Percent of Warn./Interv.	Percent of Warn./Total Warn.
December	61	8	13.1%	40.9%
January	21	19	90.5%	14.1%
February	15	2	13.4%	10.1%
March	10	3	30.0%	6.7%
April	8	3	37.5%	5.4%
May	13	3	23.1%	8.7%
June	21	0	0.0%	14.1%
Total	149	38	n/a	n/a

9.3. Observations

Purdue's semesters run from August through December in the fall, and January through May in the Spring. There are summer sessions beginning in May and June. It is in August and January that the greatest number of accounts are created, for use by classes. These accounts typically expire at the end of their semester. There are several interesting things to note about the above data which are related to our academic calendar. First is the initial surge of warning letters in December. This was somewhat expected, since no one had been running COPS regularly until December, and so many accounts with problems were probably just sitting idle. It is also interesting to note that the warning letters were highest in January, steadily decreasing until just about the end of the semester, when they took a short jump. The number of interventions also seems to start high in the beginning of the semester and then drop off radically. One plausible explanation for the sudden drop in warning letters is that our one-semester customers gain more UNIX knowledge as they progress through their classes, reducing the number of accidents. This theory might also explain why the number of interventions drops as the year presses on. Because of the high customer turnover involved in our academic computing, it seems plausible that we will see a periodic effect, as new students are given accounts and begin to learn their way around our systems. Clearly, one semester's worth of data is not enough to make any concrete conclusions about the long-term effect *chkacct* has on site and customer security. However I hypothesize that this pattern will continue: large jumps in warning letters during the first month of the semester, with the numbers waning off as the semester progresses.

The warning letter data that was collected was done only out of practicality, to prevent our staff from sending two letters to the same individual. Because no data was kept before the existence of *chkacct*, we have no way of knowing if a significant amount of novices learned about UNIX security solely because of *chkacct*. Finally, the reason for the staff intervention was not noted. It could have been because either the customer did not read the mail, or because they did not understand it.

10. Customer Response To *chkacct* and Warning Letters

Apart from a handful of thank-you letters from our customers, we heard virtually no response from our customers about *chkacct*. Usually the thank-you letters, which were not archived, consisted of appreciation for informing them of the problems in their accounts. There was one interesting complaint letter, in which the customer, after receiving a warning letter about world writable login files in his account, replied to me saying that neither he nor his lab T.A. had heard of either *chkacct* or me. He added that since it was the end of the semester his account was going away anyway and he did not care what happened to it. This is the only negative reaction I have seen so far and it probably could have been avoided with a bit more education.

11. Conclusions

Chkacct has an important limitation to be aware of: it is only useful on a system where the security of the "super-user" has not been compromised. Therefore, the security of the "super-user" is necessary for *chkacct* to be of any use. This seems to imply that *chkacct* will never be useful, but that is not the case. Indeed, file permissions are altogether useless when the "super-user" has been compromised, and therefore not just *chkacct* but many other issues are then moot. Many of the problems seen here at Purdue with regards to account security are easily solved with *chkacct* in that they are often a result of a lack of knowledge on the part of the customer. *Chkacct* performs a large part of the time-intensive work of educating the customer and fixing the problem. We will continue to recommend that our customers use it.

12. Availability

The *chkacct* package is now distributed with COPS. However, one may pick up the most recent version of *chkacct* via anonymous ftp at cc.purdue.edu (128.210.9.2) in *pub/chkacctv1.1.tar.Z.*

BIBLIOGRAPHY

Peter G. Neumann, Donn B. Parker, "A Summary of Computer Misuse Techniques," *Proceedings of the 12th National Computer Security Conference*, October 10-13, 1989.

W.V. Maconachy, Ph. D., "Computer Security Education, Training, and Awareness: Turning a Philosophical Orientation into a Practical Reality," *Proceedings of the 12th National Computer Security Conference*, October 10-13, 1989.

Dennis F. Poindexter, "Security Awareness: Making It Happen," *Proceedings of the 11th National Computer Security Conference*, October 17-20, 1988.

Dorothy E. Denning, Peter G. Neumann, Donn B. Parker, "Social Aspects of Computer Security," *Proceedings of the 10th National Computer Security Conference*, September 21-24, 1987.

Daniel Farmer and Eugene H. Spafford, "The COPS Security Checker System," *Proceedings of the Summer Usenix Conference*, pp. 165-170, June, 1990.