The COPS Security Checker System*

Purdue University Technical Report CSD-TR-993

Daniel Farmer
Computer Emergency Response Team
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
df@sei.cmu.edu

Eugene H. Spafford
Software Engineering Research Center
Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907–1398
spaf@cs.purdue.edu

January 22, 1994

Abstract

In the past several years, there have been a large number of published works that have graphically described a wide variety of security problems particular to UNIX. Without fail, the same problems have been discussed over and over again, describing the problems with SUID (set user ID) programs, improper file permissions, and bad passwords (to name a few). There are two common characteristics to each of these problems: first, they are usually simple to correct, if found; second, they are fairly easy to detect.

Since almost all systems have fairly equivalent problems, it seems appropriate to create a tool to detect potential security problems as an aid to system administrators. This paper describes one such tool: COPS. (Computerized Oracle and Password System) is a freely-available,

* This paper originally appeared in the proceedings of the Summer Usenix Conference, 1990, Anaheim CA.

reconfigurable set of programs and shell scripts that enable system administrators to check for possible security holes in their systems.

This paper briefly describes the system. Included are the underlying design goals, the functions provided by the tool, possible extensions, and some experiences gained from its use. We also include information on how to obtain a copy of the initial COPS release.

1 Introduction

The task of making a computer system secure is a difficult one. To make a system secure means to protect the information from disclosure; protecting it from alteration; preventing others from denying access to the machine, its services, and its data; preventing degradation of services that are present; protecting against unauthorized changes; and protecting against unauthorized access.

To achieve all these security goals in an actual, dynamic environment such as that presented by most UNIX [1] systems can be a major challenge. Practical concerns for flexibility and adaptability render most formal security methods inapplicable, and the variability of system configuration and system administrator training make "cookbook" methods too limited. Many necessary security administration tasks can be enhanced through the use of software and hardware mechanisms put in place to regulate and monitor access by users and user programs. Those same mechanisms and procedures, however, constrain the ability of users to share information and to cooperate on projects. As such, most computer systems have a range of options available to help secure the system. Choosing some options allows enhanced sharing of information and resources, thus leading to a better collaborative environment, where other settings restrict that access and can help make the system more secure.

One of the tasks of a system and security administrator is to choose the settings for a given system so that security is at an appropriate level—a level that does not unduly discourage what sharing is necessary for tasks to be accomplished, but that also gives a reasonable assurance of safety. This often leads to problems when a system has a very wide range of possible settings, and when system administrators lack sufficient training and experience to know what appropriate settings are to be applied.

Ideally, there should be some kind of assistance for system administrators

[1] UNIX is a registered trademark of AT&T Technologies.

that guides them in the application of security measures appropriate for their environment. Such a system needs to be configurable so it provides the appropriate level of assistance based on the perceived need for security in that environment. That system should be comprehensive enough so that an untrained or inexperienced administrator is able to derive a high degree of confidence that all appropriate features and weaknesses are identified and addressed.

Unfortunately, such a tool may also present a danger to that same system administrator. For instance, there could be a danger if the tool were to fall into the hands of a potential attacker. The tool could be used to analyze the target system or to provide clues for methods of attack. A second potential danger is that the tool can be modified by an unfriendly agent so that the information it reports and the actions that it takes serve not to enhance the security of the system, but to weaken it. A third possibility is that the tool is not comprehensive enough, or that changes in system operation are such that the tool does not expose the security flaws made present by those changes; the security administrator, by relying on the tool, fails to be aware of the new dangers to his or her system.

A good example of all three dangers might be the development and use of a tool that examines passwords to see if they can be easily guessed by an attacker. Such a tool might consist of a fast implementation of the password encryption algorithm used on a particular machine. Provided with this tool would be a dictionary of words that would be compared against user passwords. Passwords that match a word in the dictionary would be flagged as weak passwords.

Such a tool would enable a system administrator to notify users with weak passwords that they should choose a password that is more difficult for an attacker to guess. However, such a tool is a danger to the very same system it is designed to protect should it fall into the hands of an attacker: the tool could be used to very rapidly search through the dictionary in an attempt to find a password that could be compromised.

A second potential danger is that an attacker with sufficient privilege might alter the encryption algorithm or the internal workings of the program such that it would appear to run correctly, but would fail to match certain passwords or certain accounts. This would allow a determined attacker to plant an account with a known simple password that would not be detected by the program. Alternatively, an attacker might modify such a program to send its output to not only the administrator, but to the attacker as well.

The third problem is that the system administrator may grow complacent by running this password tool if it continually reports that there are no weak passwords found. The administrator may not make any effort to enhance the quality or size of the dictionary, or to provide other tracking or audit mechanisms to observe individuals who may be attempting to guess passwords or break into accounts.

For all of these reasons, such a tool might be considered to lessen the overall security of the system rather than to enhance it. That should not prevent us from developing security tools, however. Instead, the challenge is to build tools that enhance security without posing too great a threat when employed by an enemy.

2 Design and Structure

2.1 Design

Although there is no reasonable way that all security problems can be solved on any arbitrary system, administrators and systems programmers can be assisted by a software security tool. COPS is an attempt to address as many potential security problems as possible in an efficient, portable, and above all, in a reliable and safe way. The main goal of COPS is one of prevention; it tries to anticipate and eliminate security problems by detecting problems and denying enemies an opportunity to compromise security in the first place.

The potential security hazards that COPS checks for were selected from readings of a variety of security papers and books (see the references section at the end of the paper), from interviews with experienced system administrators, and from reports of actual system breakins.

We applied the following important guiding principles to the design and development of COPS:

- COPS should be configurable so that new tools could be added or the existing tools altered to meet the security needs of the installation on which it is run. Since UNIX is so dynamic, it must be possible to incorporate both new tools and methods in COPS as the need for them becomes apparent.

- COPS should contain no tool that attempts to fix any security problems that are discovered. Because COPS makes no modifications to the system, it is not required that it be run with any particular privilege, and many of the tools can be run with privilege less than or equal to that of a regular user. As a result, this lessens the temptation for an intruder to modify the code in an attempt to make surreptitious changes to the system.

- While COPS should notify the administrator that there may be a weakness, it does not describe why this is a problem or how to exploit it. Such descriptions should be found in alternative sources that are not embedded in the program. Thus, a determined attacker might run the program, might be able to read the output, but be unaware of a method to exploit anything that COPS reports it has found.

- COPS should not include any tools whose use by determined attackers, either standalone or as part of the COPS system, would give them a *significant* advantage at finding a way to break into the system beyond what they might already have in their possession. Thus, a password checking tool, as was previously described, is included, but the algorithm utilized is simply what is already present in the system library of the target system.

- COPS should consist of tools and methods that are simple to read, understand, and to utilize. By creating the tools in such a manner, any system administrator can read and understand the system. Not only does this make it easier to modify the system for particular site needs, but it allows reexamination of the code at any time to ensure the absence of any Trojan horse or logic bomb.

- The system should not require a security clearance, export license, execution of a software license, or other restriction on use. For maximum effectiveness, the system should be widely circulated and freely available. At the same time, users making site-specific enhancements or including proprietary code for local software should not be forced to disclose their changes. Thus, COPS is built from new code without licensing restrictions or onerous "copyleft," and bears no restriction on distribution or use beyond preventing it from being sold as a commercial product.

- COPS should be be written to be portable to as wide a variety of UNIX systems as possible, with little or no modification.

In order to maximize portability, flexibility, and readability, the programs that make up COPS are written as simple Bourne shell scripts using common commands (awk, sed, etc.), and when necessary, small, heavily-commented C programs.

2.2 Structure

COPS is structured as a dozen sub-programs invoked by a shell script. That top-level script collects any output from the subprograms and either mails the information to the local administrator or else logs it to a file. A separate program that checks for SUID files is usually run independently because of the amount of time required for it to search through the filesystems. All of the tools except the SUID checker are not meant to be run as user root or any other privileged account.

Please note that the descriptions of the tools provided here do not contain any detailed explanation of why the tools check what they do. In most cases, the reason is obvious to anyone familiar with UNIX. In those cases where it is not obvious, the bibliographic material at the end of this paper may provide adequate explanations. We apologize if the reasons are not explained to your satisfaction, but we do not wish to provide detailed information for potential system crackers who might have our system.

These are the individual the programs that comprise COPS:

dir.check, file.chk These two programs check a list of directories and files (respectively) listed in a configuration file to ensure that they are not world-writable. Typically, the files checked would include */etc/passwd, /.profile, /etc/rc*, and other key files; directories might include */, /bin, /usr/adm, /etc* and other critical directories.

pass.chk This program searches for and detects poor password choices. This includes passwords identical to the login or user name, some common words, etc. This uses the standard library crypt routine, although the system administrator can link in a faster version, if one is available locally.

group.chk, passwd.chk These two tools check the password file (*/etc/passwd* and yppasswd output, if applicable) and group file (*/etc/group* and ypgroup output, if applicable) for a variety of problems including blank lines, null passwords, non-standard field entries, non-root accounts with uid=0, and other common problems.

cron.chk, rc.chk These programs ensure that none of the files or programs that are run by cron or that are referenced in the */etc/rc** files are world-writable. This protects against an attacker who might try to modify any programs or data files that are run with root privileges at the time of system startup. These routines extract file names from the scripts and apply a check similar to that in file.chk.

dev.chk checks */dev/kmem, /dev/mem*, and file systems listed in */etc/fstab* for world read/writability. This prevents would-be attackers from getting around file permissions and reading/writing directly from the device or system memory.

home.chk, user.chk These programs check each user's home directory and initialization files (*.login, .cshrc, .profile*, etc) for world writability.

root.chk This checks root startup files (e.g., */.login, /.profile*) for incorrect umask settings and search paths containing the current directory. This also examines */etc/hosts.equiv* for too much accessibility, and a few miscellaneous other tests that do not fit anywhere else.

suid.chk This program searches for changes in SUID file status on a system. It needs to be run as user root for best results. This is because it needs to find all SUID files on the machine, including those that are in directories that are not generally accessible. It uses its previous run as a reference for detecting new, deleted, or changed SUID files.

kuang The U-Kuang expert system, originally written by Robert W. Baldwin of MIT. This program checks to see if a given user (by default, root) is compromisable, given that certain rules are true.

It is important to note once again that COPS does not attempt to correct any potential security hazards that it finds, but rather reports them to the administrator. The rationale for this is that is that even though two sites may have the same underlying hardware and version of UNIX, it does not mean that the administrators of those sites will have the same security concerns. What is standard policy at one site may be an unthinkable risk at another, depending upon the nature of the work being done, the information stored on the computer, and the users of the system. It also means that the COPS system does not need to be run as a privileged user, and it is less likely to be booby-trapped by a vandal.

3 Usage

Installing and running COPS on a system usually takes less than an hour, depending on the administrator's experience, the speed of the machine, and what options are used. After the initial installation, COPS usually takes a few minutes to run. This time is heavily dependent on processor speed, how many password checking options are used, and how many accounts are on the system.

The best way to use COPS is to run it on a regular basis, via at or cron. Even though it may not find any problems immediately, the types of problems and holes it can detect could occur at any later time.

Though COPS is publically accessible, it is a good idea to prevent others from accessing the programs in the toolkit, as well as seeing any security reports generated when it has been run. Even if you do not think of them as important, someone else might use the information against your system. Because COPS is configurable, an intruder could easily change the paths and files that it checks, thus making any security checks misleading or worthless. You must also assume intruders will have access to the same toolkit, and hence access to the same information on your security problems. Any security decisions you make based on output from COPS should reflect this. When dealing with the security of your system, caution is never wasted.

4 Experience and Evaluation

This security system is not glamorous—it cannot draw any pictures, it consists of a handful of simple shell scripts, it does not produce lengthy, detailed reports, and it is likely to be of little interest to experienced security administrators who have already created their own security toolkits. On the other hand, it has proven to be quite effective at pointing out potential security problems on a wide variety of systems, and should prove to be fairly valuable to the majority of system administrators who don't have the time to create their own system. Some administrators of major sites have informed us that they are incorporating their old security checks into COPS to form a unified security system.

COPS has been in formal release for only a few months (as of January 1990). We have received some feedback from sites using the system, including academic, government and commercial sites. All of the comments about the ease of use, the readability of the code, and the range of things checked by the system have been quite positive. We have also, unfortunately, had a few reports that COPS may have been used to aid in vandalizing systems by exposing ways to break in. In one case, the vandal used COPS to find a user directory with protection modes 777. In the other case, the vandal used COPS to find a writable system directory. Note, however, that in both of these cases, the same vulnerability could have easily been found without COPS.

It is interesting to note that in the sites we have tested, and from what limited feedback we received from people who have utilized it, over half the systems had security problems that could compromise the root user. Whether that can be generalized to a larger population of systems is unknown; part of our ongoing research is to determine how vulnerable a typical site may be. Even machines that have come straight from the vendor are not immune from procedural security problems. Critical files and directories are often left world-writable, and configuration files are shipped so that any other machine hooked up to the same network can compromise the system. It underscores this sad state of affairs when one vendor's operational manual harshly criticizes the practice of placing the current directory in the search path, and then in the next sentence states "Unfortunately, this safe path isn't the default." [2]

We plan on collecting further reports from users about their experiences with COPS. We would encourage readers of this paper who may use it to inform us of the performance of the system, the nature of problems indicated by the system, and of any suggestions for enhancing the system.

5 Future Work

From the beginning of this project, there have been two key ideas that have helped focus our attention and refine our design. First, there is simply no reasonable way for us to write a security package that will perform every task that we felt was necessary to create a truly satisfactory security package. Second, if we waited, no one else was going to write something like COPS for us. Thus, we forged ahead with the design and construction of a solid, basic security package that could be easily expanded. We have tried to stress certain important principles in the design of the system, so that the

[2] We will not embarrass that one vendor by citing the source of the quote. At least they noted the fact that such a path is a hazard; many vendors do not even provide that much warning.

expansion and evolution of COPS will continue to provide a workable tool.

COPS was written to be rewritten. Every part of the package is designed to be replaced easily; every program has room for improvement. The framework has room for many more checks. It seems remarkable that a system as simple as this finds so many flaws in a typical installation! Nonetheless, we have thought of a number of possible extensions and additions to the system; these are described in the following sections.

5.1 Detecting known bugs

This is a very difficult area to consider, because there are an alarming number of sites (especially commercial ones) without the source code that is necessary to fix bugs. Providing checks for known bugs might make COPS more dangerous, thus violating our explicit design goals. At the same time, checking for known bugs could be very useful to administrators at sites with access to source code.

If we keep in mind that COPS is intended as a system for regular use by an administrator, we conclude that checking for known bugs is not appropriate, because such checks are ordinarily done once and not repeated. Thus, a separate system for checking known bugs would be appropriate—a a system that might be distributed in a more controlled manner. We are currently considering different methods of distributing such a system.

5.2 Checksums and Signatures

Checksums and cryptographically-generated signatures could be an excellent method of ensuring that important files and programs have not been compromised. COPS could be enhanced to regenerate these checksums and compare them against existing references. To build this into COPS will require some method of protecting both the checksum generator and the stored checksums, however. It also poses the problem that system administrators might rely on this mechanism too much and fail to do other forms of checking, especially in situations where new software is added to the system.

5.3 Detecting changes in important files

There are some files that should change infrequently or not at all. The files involved vary from site to site. COPS could easily be modified to check these files and notify the system administrator of changes in contents or modification times. Again, this presents problems with the protection of the reference standard, and with possible complacency.

5.4 NFS and Yellow Pages

Many new vulnerabilities exist in networked environments because of these services. Their recent development and deployment mean that there are likely to be more vulnerabilities and bugs present than would be found in more mature code. As weaknesses are reported, corresponding checks should be added to the COPS code.

5.5 Include UUCP security checks

Because UUCP is very widely used, it is important to increase the number and sophistication of the checks performed on all the different varieties of UUCP. This includes checking the files that limit what programs can be remotely executed, the *USERFILE* and *L.sys* files, and the protections on directories.

5.6 Configuration files

There are many problems that result from improper configuration files. These occur not only from having the files open to modification, but because of unexpected or misunderstood interactions of options. Having rule-based programs, similar to kuang, which analyze these configuration files would be an ideal way to extend COPS.

5.7 Checking OS-specific problems

There are a wide variety of problems that apply only to certain flavors of UNIX. This includes not only the placement of key files, but also syntactical and logical differences in the way those systems operate. Examples include such things as shadow password files, different system logging procedures, shared memory, and network connectivity. Ideally, the same set of tools would be used on every system, and a configuration file or script would resolve any differences.

6 Conclusions

Over the last 18 months since the Internet worm, perhaps the most strongly voiced opinion from the Internet community has been "security through secrecy does not work." Nonetheless, there is still an appalling lack of communication about security. System breakers and troublemakers, on the other hand, appear to encounter little difficulty finding the time, energy, and resources necessary to break into systems and cause trouble. It is not that they are particularly bright; indeed, examining the log of a typical breakin shows that they follow the same methods that are publicized in the latest computer security mailing lists, in widely publicized works on security, and on various clandestine bulletin boards. The difference between them and the system administrators on the Internet seems to be communication. It is clear that the underground community has a well-established pipeline of information that is relatively easy for them to tap. Many system administrators, however, have no access to an equivalent source of information, and are thrust into their positions with little or no security experience. Cops should be particularily helpful in these cases.

None of programs in Cops cover all of the possible areas where a system can be harmed or compromised. It can, however, aid administrators in locating some of the potential trouble spots. Cops is not meant to be a panacea for all Unix security woes, but an administrator who examines the system and its documentation might reduce the danger to his or her system. That is all that can ever be expected of any security tool in a real, operational environment.

Future work on Cops will be done at the CERT, and work on related tools and approaches will be done at Purdue. People are encouraged to get a copy of Cops and provide us with feedback and enhancements. We expect that as time goes on, and as the awareness of security grows, Cops and systems like it will be evolved through community effort. Increased communication and awareness of the problems should not be limited to just the crackers.

7 Acknowledgments

Thanks go to Robert Baldwin for allowing us to include his marvelous U-Kuang system; to Donald Knuth for inspirational work on how not only to write but to create a software system; to Jeff Smith, Dan Trinkle, and

Steve Romig for making available their systems and expertise during the development of COPS; and finally, our beta testers, without whom COPS might never have been.

Getting COPS

COPS has been run successfully on a large number of computers, including UNIX boxes from Sun, DEC, HP, IBM, AT&T, Sequent, Gould, NeXT, and MIPS.

A copy of COPS was posted to the *comp.sources.unix* newsgroup and thus is available in the UUCP archives for that group, as well as via anonymous ftp from a variety of sites (*uunet.uu.net* and *j.cc.purdue.edu*, for example.) We regretfully cannot mail copies of COPS to sites, or make tapes, as we do not have the time or resources to handle such requests.

Biographies

Dan Farmer is a member of the CERT (Computer Emergency Response Team) at the Software Engineering institute at Carnegie Mellon University. He is currently designing a tool that will detect known bugs on a variety of UNIX systems, as well as continuing program development and design on the UNIX system.

Gene Spafford is an assistant professor at Purdue University in the Department of Computer Sciences. He is actively involved with software engineering research, including testing and debugging technology. He is also actively involved in issues of computer security, computer crime, and professional ethics. Spaf is coauthor of a recent book on computer viruses, is in the process of coauthoring a book on UNIX security to be published by O'Reilly and Associates, and is well-known for his analysis of the Morris Internet Worm. Besides being a part-time netgod, Gene is involved with ACM, IEEE-CS, the Computer Security Institute, the Research Center on Computers and Society, and (of course) Usenix.

References

1. Aho, Alfred V., Brian W. Kernighan, and Peter J. Weinberger, *The AWK Programming Language,* Addison-Wesley Publishing Company, 1988.

2. Authors, Various, UNIX *Security Mailing List/Security Digest,* December 1984-present.
3. Baldwin, Robert W., *Rule Based Analysis of Computer Security,* Massachusetts Institute of Technology, June 1987.
4. Grampp, F. T. and R. H. Morris, "UNIX Operating System Security," *AT&T Bell Laboratories Technical Journal*, October 1984.
5. Kaplilow, Sharon A. and Mikhail Cherepov, "Quest—A Security Auditing Tool," *AT&T Bell Laboratories Technical Journal,* AT&T Bell Laboratories Technical Journal, May/June 1988.
6. Smith, Kirk, "Tales of the Damned," UNIX *Review,* February 1988.
7. Spafford, Eugene, Kathleen Heaphy and David Ferbrache, *Computer Viruses: Dealing with Electronic Vandalism and Programmed Threats,* ADPASO, 1989.
8. Spence, Bruce, "spy: A UNIX File System Security Monitor," *Proceedings of the Large Installation Systems Administration III Workshop,* Usenix Association, September, 1988.
9. Thompson, Ken, "Reflections on Trusting Trust," **27** (8), *Communications of the ACM,* August 1984.
10. Wood, Patrick and Stephen Kochran, UNIX *System Security,* Hayden Books, 1986.
11. Wood, Patrick, "A Loss of Innocence," UNIX *Review,* February 1988.