The following paper was originally published in the
Proceedings of the Sixth USENIX UNIX Security Symposium
San Jose, California, July 1996.

NetKuang - A Multi-Host Configuration Vulnerability Checker

Dan Zerkle and Karl Levitt
Department of Computer Science
University of California at Davis

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

NetKuang – A Multi-Host Configuration Vulnerability Checker

Dan Zerkle and Karl Levitt
Department of Computer Science
University of California at Davis

zerkle@cs.ucdavis.edu, levitt@cs.ucdavis.edu

Abstract

NetKuang is an extension to Baldwin's SU-Kuang. It runs on networks of computers using Unix and can find vulnerabilities created by poor system configuration. Vulnerabilities are discovered using a backwards goal-based search that is breadth-first on individual hosts and parallel when multiple hosts are checked. An implementation in C++ found real vulnerabilities on production systems. Tests show reasonably fast performance on an LAN.

1 Introduction

The security of modern networked computer systems is dependent on more than just the integrity of the software and protection mechanisms their operating systems use; it is also dependent on the proper configuration and use of that software. Unix computers have a wide variety of security mechanisms such as file permissions, passwords, trusted hosts, and so forth. In practice, such mechanisms can quickly grow very complex. A simple configuration error can lead to users gaining unintended access. The problem has grown worse with the popularity of networked systems. There are more hosts to configure, and the security is dependent on more mechanisms.

This paper presents NetKuang, a system which addresses some of these security concerns by checking networked configurations for unintended security vulnerabilities.

§2 reviews existing systems to enhance Unix security and justifies the need for NetKuang. §3 presents the design of our NetKuang system. It considers the basic functionality of the system, the algorithms used, and the limitations of the design. §4 discusses the current prototype implementation. §5 presents a detailed discussion of the search technique used by the prototype. §6 considers planned enhancements. §7 presents the results of some experiments with this tool. Finally, §8 presents our conclusions about the initial prototype of NetKuang.

2 Previous Systems

A number of configuration analysis tools have been developed to check whether a system is vulnerable to attack based on the content of system tables. Given that is very likely that an administrator will make mistakes in configuring his system and that many of these mistakes can leave the system open to easy attack, these tools have been widely used as a preventive measure.

The Computer Oracle Password and Security System (COPS) [1], uses a set of shell scripts to check for likely misconfigurations in a Unix system. Among its simple but important checks are the permission modes of security-relevant files and directories, such as `/etc/passwd` and `/etc/group`. COPS also determines if set user-ID root files are world-writable.

The SU-Kuang system [2] is a rule-based expert system for checking the security of a Unix file system's configuration. The rule base captures approaches by which an attacker can extend his privileges by making system calls. Examples of such rule are "if a user can write to a directory, then the user can replace any file inside the directory" and "a user who can replace the password file is able to acquire superuser privileges." Using these rules, SU-Kuang exhaustively searches for all possible actions that enable a user to acquire privileges inconsistent with a specified policy. Although very effective for single host Unix systems, it does not work on a network of Unix systems.

The Miro security constraint file checking system [3] checks a file system against a set of specified constraints that define legal configurations of the system. Security constraints are specified as graphs,

and the system administrator uses a graphical editor to create and modify graphs. A constraint checker determines if the graphs are consistent with the policy.

Tripwire [4], the widely used file integrity checker, is used to determine if critical files have been modified. An attacker might write trojan horses or trapdoors into critical programs; for example into the `login` program to allow him future access to the system without presentation of password, or into `ls` or `ps` so that calls to these programs will not reveal the presence of modified files or processes. In Tripwire, a cryptographic checksum is computed for each file, and the original checksum is compared periodically with that for the current version of the file.

The Security Administrator's Tool for Analyzing Networks (SATAN) [5] and the Internet Scanner [6] both scan networks to find vulnerable hosts. They can look for such suspicious states as use of faulty versions of network software and improper NFS exports.

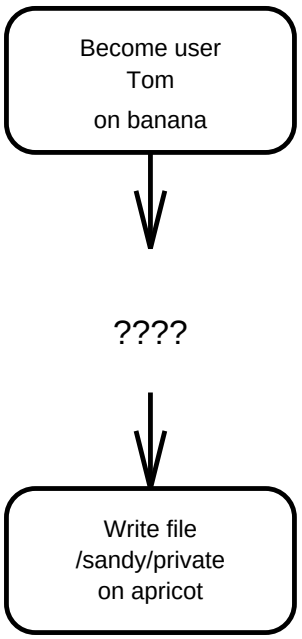

Figure 1: An example query. Given the starting privileges, is it possible to gain the ending privileges?

3 NetKuang

Netkuang is based on Baldwin's SU-Kuang [2], It has all the functionality of SU-Kuang, but also addresses the concerns of a networked environment. It is capable of searching a large number of hosts in parallel, and it also considers potential configuration vulnerabilities present in a networked environment.

NetKuang and SU-Kuang are named after a fictional piece of security-breaking software in William Gibson's novel, *Neuromancer* [7].

3.1 Functionality

An example will best illustrate the kind of problem that NetKuang can find. Consider a user *Sandy*, who is logged into a Unix machine named *apricot.* She wants to find if someone logged into *Tom's* account on host *banana* can potentially modify a file in her home directory called `private`. This possibility is illustrated in Figure 1.

NetKuang might find the following: A user named *Larry* has an account on banana, and his home directory permissions are set to be group-writable. Larry and Tom are both members of the *staff* group, so Tom could replace Larry's shell startup script file. The next time Larry logs in, the modified script makes a copy of the shell. The copy is owned by Larry's account, and has set-user-ID permissions set. Therefore, Tom can log in to banana as Larry.

Another host, *peach,* has an account owned by Larry. Also, peach has a `/etc/hosts.equiv` file that contains banana's name. Therefore, Larry (and now Tom) can log into peach without providing a password. Peach is an NIS server for the network, including apricot and banana. It keeps the NIS database files in a directory called `/var/yp`. The permissions for the `/var/yp` directory are accidentally left world-writable. Therefore, someone using Larry's account can replace the NIS directory with a directory full of modified NIS files. In particular, it is possible to replace the NIS password file there.

Once Tom uses Larry's account on peach to replace the NIS password file, he can give Sandy's account any password he wants. Once this is done, he can use the new password to log in to apricot using the new password. Once logged in, he can use Sandy's account to modify private.txt. This path of privileges is shown in Figure 2.

NetKuang works with privileges as goals. There are only four kinds of goals, as follows: Become (log in as) a user, become a member of a group, write to a file, and replace a file. Each goal is tied to a particular host and has an argument. For example, a fully specified goal is to `write to the file named /etc/passwd on host calvin.comics.com`.

As shown in the example above, privileges of one kind may lead to other privileges. For example, if a user can write to the `/etc` directory on some host, then he can also replace the `/etc/passwd` file on that same host. Doing that will probably lead to further privileges. NetKuang can find long chains of such

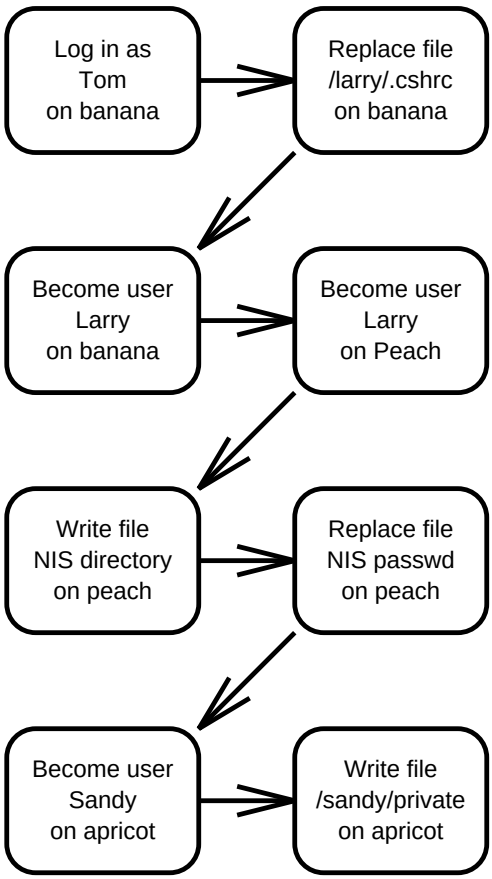

Figure 2: An example path corresponding to the above query.

privileges.

NetKuang, as SU-Kuang, considers such things as file permissions and directory structure. As in SU-Kuang, it also considers the special cases of the permissions on important files such as the password file and users' shell startup files. It has several features beyond SU-Kuang's. It considers the contents of the `/etc/hosts.equiv` file and users' `.rhosts` files to identify which other users can log in without passwords. It also considers the NIS database files, and what privileges may be gained by modifying them. Finally, NetKuang allows wild cards. It can consider all the members of certain groups of users. The set of all rules and goals used by NetKuang is listed in Appendix A.

The user supplies the desired ending goal privileges and the starting privileges that some theoretical misuser might have. NetKuang then performs a search to determine if it is possible to attain the goal privileges from the start privilege. If it finds a solution, it reports the path of privileges leading from the start to the goal.

3.2 Limitations

There are some limitations to the current NetKuang prototype.

It does not consider the integrity of various security-critical software. It does not check for bad passwords or the presence of network sniffers. It does not attempt to parse many important system configuration files, that may have security-critical contents, such as the `/etc/rc` file.

NetKuang only finds a single match. It may be that multiple paths exist between the start and goal privileges. The only way to discover further paths with NetKuang is to fix the known configuration vulnerability and run another search.

4 Current Implementation

The current implentation of NetKuang is written in Gnu C++. It runs on a small network of SunOS 4 and 5 systems. It is currently being ported to IRIX and Ultrix systems on a much larger network for further evaluation.

The search engine is implemented as a daemon on each machine that can be searched. This allows the daemon to directly examine the file system in question. Each engine is capable of participating in multiple simultaneous searches. If a search on a request fails, the engine may request other engines to carry out the search further. See §5 for a detailed description of the engine's search technique.

The user interface is a simple program that collects the start and destination goals from the user and forwards them to the host identified in the destination goal. Upon completion of the search, the user interface either reports failure or reports the path of goals from the start to the destination. It also notifies the search engines to cancel the completed search.

Communication in NetKuang is carried out by a handcrafted message passing system called ZCS, developed for wide-area auditing. A ZCS message dispatcher daemon collects all NetKuang requests and responses as UDP messages and redistributes them to the modules that need them. Further description of ZCS is not essential for this paper and will be addressed in a future publication.

The search engines must run with super-user privileges in order to examine any `.rhosts` files of users that would not otherwise be readable. If this information is not needed, it may run as any user and unreadable `.rhosts` files are, of course, ignored.

5 Search Technique

NetKuang uses a backward-chained goal-based search. Given starting privileges and ending privileges, NetKuang analyzes the systems on which it is run to determine if the starting privileges are adequate to achieve the ending privileges. The search is carried out breadth-first on each individual host, and in parallel between different hosts. This section details the search technique as it is currently implemented.

5.1 Goals

Searching by NetKuang is based on the generation of goals. Each goal represents privileges that a given user may hypothetically acquire. There are three fields for each goal. The fields are the type of the goal, an argument associated with the type, and the name of the host on which the goal might be met. The different types of goals and their corresponding arguments are listed in Table 1.

Specifying a user by ID and by name are very similar. They simply imply the ability to become the specified user. Unix systems grant most privileges to users based on the value of user ID's. However, the contents of `.rhosts` and `hosts.equiv` files may allow users to log in from remote systems without providing passwords. The identity of remote users is determined by the names of the users, not their ID numbers, so NetKuang tracks both names and ID numbers.

Goal Type	Argument
Become user by ID	User ID number
Become user by name	User name string
Become member of group	Group ID number
Write file	File's path name
Replace file	String file's path name

Table 1: Possible goals and their arguments

Replacing and writing a file are also similar in that they imply the ability to modify the contents of a file. However, a file must exist before it can be written, while even a non-existent file might be replaced. Also, the ability to write a file implies that the ownership of that file does not change upon modification. Thus, the ability to write a file implies the ability to replace it, but not the other way around. The difference between writing and replacing is important because `.rhosts` files must be owned by the appropriate user or else Unix ignores them when remote users use the `rlogin` protocol.

Here are some example goals:

- `Become user named zerkle on host krakatoa`
- `Become member of group 1 (staff) on host dino`
- `Replace file /etc/passwd on host calvin`
- `Become user ID 0 on host jade`
- `Write file /usr/home/jack/.cshrc on host trusty`

Some goals contain *wild cards*. Goals with wildcards imply special privileges. They are mainly used to specify one complicated search instead of many simple ones.

A goal with a host name of `all` implies the possession of the specified privileges on all hosts searched by a particular instantiation of NetKuang. The host name `any` in a goal implies the possesion of the privileges on at least one searched host.

The two `become user` goals use special wild cards. The user names `all` and `any` imply the ability to become all users or any user at a given host.

Wild cards exist in NetKuang mainly for two particularly interesting goals:

- `Become user all at host all`
- `Become user root at host any`

5.2 Goal Expansion

Goal expansion is the heart of NetKuang's search mechanism. Expanding a goal results in a set of new goals, any one of which, if achieved, grants the original goal privileges. The expansion answers the question, "What are the ways in which I could acquire these privileges?"

Some expansions have preconditions. For instance, the goal `write file` might expand to the ability to become any user in the group of the user that owns the file. The preconditions in this case are that the file exists and that its permissions are set to allow writing by the owner's group.

For instance, the goal

- `become user ID 0 at host apple`

might expand to (among others)

- `replace file /etc/passwd at host apple`,
- `become user named ''root'' at host apple`,
- `write file /.rhosts at host apple`,
- `replace file /var/yp/users/passwd.byuid.pag at host melon`.

As shown by this example, some of the goals resulting from an expansion can be met by the same host as the original goal, but some may be met only by another host.

Expansion of goals with wild cards is handled in a special manner. The goals with the `all` host or user wild card do not expand at all. However, every other goal, in addition to its normal expansions, expands to the same goal with `all` as its host, and every `become user` goal expands to `become user all` at the same host. Every goal, in addition to its normal expansions, expands to the same goal with `all` as its host. A goal with host `any` is expanded to the identical goal for every host on which NetKuang is running, but is not otherwise expanded. The goal of becoming `any` user at a given host expands to one such goal for every user that can log in to that host. All `become user` goals expand to becoming the `all` user on the same host. The goals with the `all` host or user wild card do not expand at all.

A complete list of the possible expansions is given in Appendix A.

5.3 Search Algorithm

A NetKuang search consists of a test of whether a given destination goal can be met from a starting privileges goal. Typically, a search is started by a user request to a search engine.

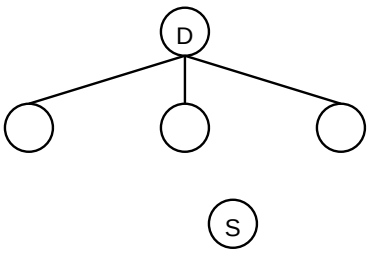

Figure 3: A local search after the first expansion. Node D is the destination goal and node S is the start goal.

NetKuang carries out both local and remote searching. Local searching consists of goal generation and analysis of the local host to determine if the destination goal can be expanded to the origin goal. Remote searching is done if local searching fails to find a solution. It consists of requests to search engines on other hosts to carry out searches to determine if they can meet the goal.

5.3.1 Local Search

Local search is a breadth-first expansion of a goal tree. It is carried out upon receipt of a search request by a search engine, where the request may be made by a user interface or by another search engine. The search request consists of a two goals. One goal identifies start privileges, and the other goal lists destination privileges. Upon receipt of the request, the destination goal is expanded to a new set of goals, as shown in Figure 3. Note that this search is *backwards*. The expansion proceeds from the destination goal in an attempt to find the start goal.

Each of the new goals from the expansion is examined to see if it is the start goal. If not, each of the new goals is expanded. Figure 4 shows these goals partially expanded. One of the new goals is the start goal. This indicates a successful search.

Different goals, when expanded, may result in some identical expansion goals. These might be represented as non-tree links in the search tree, as shown in Figure 5. Such duplicate goals must *not* be further expanded, or loops will result and the search will continue indefinitely. Therefore, the generated goals are kept in a hash table. Newly generated goals already in the table are discarded.

Goals referring to hosts not identical to the host on which the local search is carried out can not be expanded locally. Instead, they are saved for a pos-

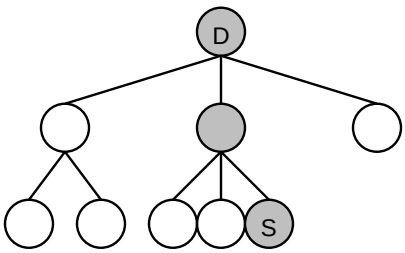

Figure 4: A local search after the second expansion. The start goal has been found. Shaded nodes indicate goals along the successful path.

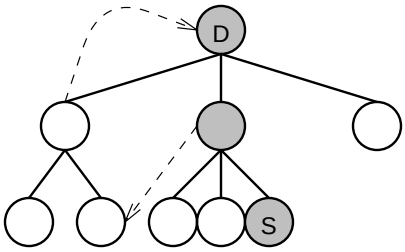

Figure 5: A local search showing non-tree expansions to duplicate goals. The dashed lines indicate the duplicate expansions.

sible remote search.

If the start goal is found, the path between the start and end goals is reported to the search engine or user interface that made the search request. Because the search is breadth-first, it will always find the shortest existing path to the start goal.

If the start goal is not found and no further non-duplicate goals can be generated, the local search is considered to be a failure. If any remote goals are saved from the local search, the search engine will start a remote search.

5.3.2 Remote Search

A search engine carries out a remote search when the local search fails.

For every incoming query, the search engine keeps a set of goals involving remote hosts (actually, there is a queue of goals for each mentioned remote host). When starting the remote search the engine sends one request to each remote host mentioned in a goal generated in the local search. These goals are all sent immediately. The search engine does not wait for results. This means that all the hosts queried can search in parallel, dramatically speeding the search over a serial approach.

If any of the remote requests returns a successful goal path, the engine appends the remote path to the local path used to reach the original remote goal, then returns this successful result to the the user interface or search engine that originated the incoming query.

If a remote request reports a failed search and the incoming query still has remote goals mentioning the reporting host, the search engine will send another query to that host. If no remote goals are left, the engine reports failure to the host that generated the incoming query. This also happens if the local search stage generates no new remote goals.

As with the local search, this recursive remote querying may generate duplicate goals. Therefore, for each ongoing search, each search engine keeps the hash table of previously generated goals. Goals generated from a previous query but the same overall search are ignored in subsequent expansions. A separate hash table is kept for each ongoing search, so the same search engines may be used simultaneously by different searches.

6 Planned Enhancements

Several enhancements are underway to the current implementation of NetKuang.

NetKuang will consider NFS file equivalency. It is possible to modify an NFS-mounted file that is mounted on one host if it is possible to write to the same file on another host. It is quite difficult to keep track of where files are mounted. The planned algorithm is to consider only the host serving a particular remote-mounted file. If the file is local and exportable (the local machine is an NFS server), then the goal of writing the file on a local system expands to writing it on all systems to where it is exported.

Although duplicate searches are avoided on any one particular host, they are not avoided between hosts. If a request included intermediate and failed goals, it would allow hosts to avoid duplication at the expense of higher communication costs. An experiment will be done to determine the performance impact of this feature.

In actual deployment, most search requests are the generic request: "Can any user on any machine get super-user privileges on any machine?" As such, it is a good idea to save results of previous searches and use them to speed up subsequent searches. In fact, the search for super-user privileges could be done on each machine as it starts up its search engine. The partial results are mostly in the form of failed goals which should not be checked again. However, once part of a host's file system changes, some of those goals become invalid and must be discarded.

As any intermediate goal may be generated by multiple paths, it is not clear which such goals should be discarded and which may be kept. An attempt will be made to implement such incremental search.

7 Experimental Results

NetKuang was deployed and run for experimental purposes on a network of ten Sun Sparc workstations. Performance was reasonably fast. It found some real configuration errors.

7.1 Sample Runs

This section details performance tests done with NetKuang. Each test was run five times, and the mean processing time was calculated for each test.

7.1.1 Error on the NIS Server

This test conjectures a configuration error on `olympus`, an NIS server. The error does not actually exist, but the start condition implies it. This demonstrates the "what if" capabilities of NetKuang. Note that the path found was *not* the shortest possible one. The results of a parallel search are nondeterministic, and the first successful goal path is the one reported. The mean run time was 6.2 seconds.

Start condition:

- `Write file /etc at olympus.cs.ucdavis.edu`

End goal:

- `Write file /home/zerkle/tmp at krakatoa.cs.ucdavis.edu`

Goal path:

1. `Write file /etc at olympus.cs.ucdavis.edu`
2. `Replace file /etc at olympus.cs.ucdavis.edu`
3. `Replace file /etc/passwd at olympus.cs.ucdavis.edu`
4. `Become user number 0 (root) at olympus.cs.ucdavis.edu`
5. `Write file /var/yp/seclab.cs/passwd.byuid.pag at olympus.cs.ucdavis.edu`
6. `Replace file /var/yp/seclab.cs/passwd.byuid.pag at olympus.cs.ucdavis.edu`
7. `Become user number 494 (zerkle) at lhotse.cs.ucdavis.edu`
8. `Become user named zerkle at lhotse.cs.ucdavis.edu`
9. `Become user number 494 (zerkle) at krakatoa.cs.ucdavis.edu`
10. `Write file /home/zerkle/tmp at krakatoa.cs.ucdavis.edu`

7.1.2 Can I get root on the server?

This query starts a single-host search on the department server. It is roughly equivalent to an SU-Kuang search, except that it may be initiated from anywhere the NetKuang user interface can be run. Fortunately, the search failed. The mean run time is 1.3 seconds.

Start condition:

- `Become user named ``zerkle'' at toadflax.cs.ucdavis.edu`

End goal:

- `Become user named ``root'' at toadflax.cs.ucdavis.edu`

7.1.3 Can anybody get root anywhere?

This is probably the most standard test. It basically asks if any user on any machine can get root on any machine. The fault found is that the `/etc` directory on lhotse is not owned by root. This run shows a minor limitation of NetKuang. The user interface, running on krakatoa, translated user number 2 to "sys". However, on lhotse, user number 2 is actually "bin". The average run time was 2.9 seconds.

Start condition:

- `Become user named ``all'' at all`

End goal:

- `Become user named ``root'' at any`

Goal path:

1. `Become user named all at all`
2. `Become user named all at lhotse.cs.ucdavis.edu`

3. `Become uid ALL USERS (non-root) at lhotse.cs.ucdavis.edu`
4. `Become user number 2 (sys) at lhotse.cs.ucdavis.edu`
5. `Write file /etc at lhotse.cs.ucdavis.edu`
6. `Replace file /etc at lhotse.cs.ucdavis.edu`
7. `Replace file /etc/passwd at lhotse.cs.ucdavis.edu`
8. `Become user number 0 (root) at lhotse.cs.ucdavis.edu`
9. `Become user named root at lhotse.cs.ucdavis.edu`
10. `Become user named root at any`

7.1.4 Can I get root anywhere?

The previous run found a solution too quickly. This run forced the system to check for root-granting configurations on the whole system as a performance check. This search failed. The mean run time was 14.3 seconds.

Start condition:

- `Become user named ''zerkle'' at all`

End goal:

- `Become user named ''root'' at any`

7.1.5 Can he log in as me?

This test run is for the sole purpose of demonstrating rlogin privileges granted though the `.rhosts` files. For this run, a vulnerability was purposely introduced to allow user `puketza` on host k6 to log into the `zerkle` account on toadflax. The mean run time was 3.0 seconds.

Start condition:

- `Write file /home/puketza/.cshrc at k6`

End goal:

- `Write file /home/zerkle/tmp at toadflax`

Goal path:

1. `Write file /home/puketza/.cshrc at k6.cs.ucdavis.edu`
2. `Replace file /home/puketza/.cshrc at k6.cs.ucdavis.edu`
3. `Become user number 423 (puketza) at k6.cs.ucdavis.edu`
4. `Become user named puketza at k6.cs.ucdavis.edu`
5. `Become user number 494 (zerkle) at toadflax.cs.ucdavis.edu`
6. `Write file /home/zerkle/tmp at toadflax.cs.ucdavis.edu`

7.2 Errors Found

During the testing of NetKuang, some configuration errors were found.

One machine contained several unexpected entries in its t/`.rhosts` file. These entries allow root access to the vulnerable machine to all hosts mentioned in the file.

On the department mail server, the directory containing the NIS database was left world-writable, which means that any user could get root access by replacing the NIS password file.

8 Conclusions

NetKuang is a tool for network administrators that has already demonstrated its usefulness by finding a serious configuration vulnerability. By considering complex configurations and large numbers of hosts, it helps secure modern networks of Unix hosts.

Future plans for NetKuang include integration into an automated intrusion detection system. It can be run periodically to determine if any new vulnerabilities have appeared. If so, they may indicate the presence of an intruder. After an intruder has been detected, it should be run immediately to determine if the intruder has created new vulnerabilities.

NetKuang is under continuing development Complete source and documentation for the latest version of NetKuang is available on the World Wide Web at `http://seclab.cs.ucdavis.edu/~ zerkle/netkuang.`

9 Acknowledgements

Thanks to Todd Heberlein and Calvin Ko for the original idea behind NetKuang.

References

[1] D. Farmer and E. H. Spafford. The cops security checker system. In *Proceedings of the Summer 1990 Usenix Conference*, June 1990.

[2] Robert W. Baldwin. Kuang: Rule-based security checking. Documentation in ftp://ftp.cert.org/pub/tools/cops/1.04/cops.104.tar.

[3] A. Heydon. Specifying and checking unix security constraints. In *Proceedings of the 3rd USENIX Security Symposium*, September 1992.

[4] Gene Kim and E. H. Spafford. The design of a system integrity monitor: Tripwire. Technical Report CSD-TR-93-071, Department of Computer Sciences, Purdue University, West Lafayette, Indiana, November 1993.

[5] D. Farmer and W. Venema. Security administrator's tool for analyzing networks. http://www.fish.com/ zen/satan/satan.html.

[6] Internet Security Systems. Internet scanner. http://www.iss.net/iss/scanner.html.

[7] William Gibson. *Neuromancer*. Ace Books, 1984.

A Search Rules

This appendix lists all of the goal expansion rules used by NetKuang. Unless specified otherwise, the expanded goals refer to the same host as the original goal.

For the purposes of this table, U refers to a user ID number and G refers to a group ID number. A full path name is specified as /d/f, where f is the last component of the path and d is the directory containing f (f may be the name of a directory). The function user(/d/f) refers to the user ID number of owner of the file, while the function group(/d/f) refers to the group ID. On the other hand, uid(name) and gid(name) refer to the user and group ID's associated with user and group names. The uname(U) function reverses this. The expression home(U) refers to the home directory of user U. "Any trusted users" refers to the set of all users specified by user name and host name that are trusted, according to the .rhosts and hosts.equiv files. "Become any user in G" expands to a separate goal for each user in the specified group. These last two may expand to many goals.

Each of these expansions is taken if the precondition is true.

In addition to the expansion in this appendix, goals with wild cards are expanded as described in §5.2.

Replace file /d/f	
Precondition	Expansions
/d/f is not the root directory	replace file /d
(none)	write file /d/f

Table 2: Expansions of `replace file` goals.

Write file /d/f	
Precondition	Expansions
/d/f exists	become user(/d/f)
/d/f exists and is group-writable	become member group(/d/f)
/d/f exists and is world-writable	become any user

Table 3: Expansions of `write file` goals.

Become user "name"	
Precondition	Expansions
(none)	become uid(name)

Table 4: Expansions of `become user name` goals.

Become user-ID U	
Precondition	Expansions
(none)	become uname(U)
(none)	replace file /etc/passwd
(none)	replace password file on NIS server
(none)	become any trusted user
file home(U)/.rhosts exists	write file home(U)/.rhosts
file home(U)/.cshrc exists	replace file home(U)/.cshrc
file home(U)/.login exists	replace file home(U)/.login
file home(U)/.profile exists	replace file home(U)/.profile
U != 0	replace file /etc/hosts.equiv
U == 0	replace file /usr/lib/crontab
U == 0	replace file /usr/lib/aliases
U == 0	replace file /etc/rc
U == 0	replace file /etc/rc.local

Table 5: Expansions of `become user ID` goals.

Become group G	
Precondition	Expansions
(none)	replace file /etc/group
(none)	replace group file on NIS server
(none)	become any user in G

Table 6: Expansions of `become member of group` goals.