Global Information Assurance Certification Paper

Password Security in NIS Systems

Eric Gallagher

July 19, 2001

SANS GIAC Security Essentials Certification Practical

Version 1.2e

Scope

This material begins with a dual survey of NIS security and password security and goes beyond the initial reading into an attempt to advance password security practice in NIS. It could present a launching point for further projects in NIS security. The necessity of these security projects is taken as given, due to the number of established sites using NIS, the new sites coming online, and the multiplatform or administrative inadequacies of some of the proposed successors to NIS.

Introduction

In many work environments of the mid 1980s and early 1990s, NIS began its life cycle as a way to ease the system maintenance workload associated with running multiple SunOS workstations. The networks of associated computers were generally designed without consideration for system security. Convenience was the goal. The same can be said of the early versions of NIS itself [1] and commercial development of NIS has essentially halted without achieving real security. To make matters worse, most offices seem to have made no allowances for migration from NIS to any other systems administration package.

Just as there is a need in those offices for migration from NIS to NIS+, LDAP, AFS, DFS, or some other secure Unix-based systems administration software, there is a need to make NIS itself more secure. Since any organization that hasn't already migrated from NIS to something more secure is likely to have a small budget, the security tools must be free or nearly so, meaning available for only a nominal fee.

Although there is a fair amount of documentation for NIS, it is surprising to find no interest group specializing in NIS security matters. In fact, the lack of detailed material is distressing from a security standpoint. There are at least two major needs that seem to go unaddressed in the current literature. This paper will deal with both in an abbreviated fashion.

One serious problem is that there is no current, recommended step-by-step algorithm or outline of the procedures needed to set up and secure NIS. (The closest is an old set of instructions released by Auburn University. [2]) This seems to be the situation despite the fact that administrators continue to deploy new NIS installations, particularly among Linux workgroups. Even administrators with the advantage of more secure systems, such as NIS+, seem to occasionally migrate back to NIS for the sake of administrative ease [3] or cross-platform compatibility. With the advent of MacOS X and its NIS (but not NIS+)

compliance [4] the problem is likely to grow. Generally, administrators seem to choose NIS for the convenience of distributing the same username, uid, and password to multiple machines and for administering NFS in a unified manner.

The compatibility issue leads into the second problem, which is the one of password aging. This may seem like a trivial issue, since it is one that should already have been dealt with by various updates to yppasswd such as linking it to passwd in Solaris. [5] However, few if any of these solutions are cross-platform and none of them appear to solve the password-aging problem in an NIS-distributed manner. Administrators who turn to NIS for its ease of use are not likely to want to enforce password rules on a machine-by-machine basis. Therefore, a way to enforce a good, secure password policy on the NIS server itself is needed.

Basic NIS Security

In order to make the NIS passwords safe (or even worth securing), it's necessary to lock down NIS in every way possible. The reasons for this have to do with the trusting nature of the original NIS administrative system. In broadcast mode, which is the default installation, all the basic maps, such as passwd, hosts, group, netgroup, mail.aliases, auto.home, and auto.master, are available to NIS clients in DBM format from the master server or any slave server. [6] This list can include custom maps. The information gets distributed freely to any NIS client listening for the right NIS domain information. The administrator may not even know about a client pretending to belong to the same NIS domain. Yet the client and the cracker running it can download all the NIS-relevant network information, including the password file, available to anyone at the command line with

% ypcat –k passwd

The password map gets passed to the NIS client (possibly a rogue client) with entries like:

jd:7mf932%90:99:Jane Doe CompName DivName Phone jdoe@compname.com:/bin/csh

which include the password hash in the second field of the record. This is true in a cross-platform situation even if shadow passwords are turned on at the server level. [7] Clearly, anyone capable of downloading password maps can run a password cracker against them and produce usable passwords in short order. Even good passwords, rotated often, will be vulnerable to such an attack. So NIS password security should begin with these steps:

1. **Install and "harden" the NIS server operating system**. This is a long and sometimes difficult process involving many hours, even multiple workdays depending on the operating system, level of security, and office environment. The SANS guides for Solaris, IRIX, and Windows NT provide step-by-step outlines to make the job reasonably quick for an experienced operator, including notes on the

importance of doing the system hardening offline. A Unix-based personal firewall such as Portsentry [8] is helpful here.

2. **Install a patched version of NIS**, the latest for the server operating system. All of the major Unix operating systems have recent patches available for download. It's likely that an NIS-related patch exists, even when installing a recent release. In any case, it's wise to install all security-related patches. The interrelation of various software tools and libraries make this necessary even when local services are restricted. For instance, NIS depends on either portmapper or rpcbind, depending on the operating system. Patches to either program must be installed or NIS will be vulnerable to known exploits against them.

3. **Choose a good NIS domain name**. (An NIS domain, like a Microsoft domain, should not be confused with an Internet domain. The name is separate and irrelevant to DNS.) The NIS domain name should be selected in the manner of a strong password. A good case can be made for changing it as often as a password. However, the change must be propagated to the NIS clients, so it might be a worthwhile project for some administrator to automate this through Expect/SSH or possibly Perl. (YP_CHDOM would be a good name for this.)

4. **Create the /var/yp/securenets file**. Thanks to patches for old operating systems, the securenets option is available for most legacy NIS systems. [9] It enables the NIS administrator to restrict map access to particular subnets or NIS clients.

5. **Install and configure tcpwrappers**, if it has not already been done. This step should have been taken in the operating system hardening phase but, if not, tcpwrappers should be configured as a prerequisite to using secure rpcbind.

6. **Load the secure version of rpcbind** and configure the hosts.allow file appropriately. This should prevent such attacks as theft of the NIS password file, ypset to force hosts to bind to a rogue NIS server, and theft of NFS file handles. The author of secure rpcbind recommends linking the RPC daemons against the securelib library [10] if it is possible on every operating system.

7. **Limit NIS user access via local passwd table entries**, where appropriate. NIS itself is capable of user-level security as far as machine and file-level access are concerned. Users should probably not be able to login to the NIS server.

8. **Further limit user access via the nsswitch.conf file**. The NIS server should probably not be its own client. If it needs to be, i.e. for the sake of automount or autofs maps, the rest of NIS access can be turned off in /etc/nsswitch.conf.

9. **Agree upon and implement an account creation policy**. This should involve decision makers beyond the local NIS administrator.

10. **Implement password expiration as a part of the account policy**. The task can be done through the steps described below on the creation of a home-grown expiration system or through the installation of an alternative password tool, some of which may support both expiration times and NIS compliant client-server password exchanges.

11. In maintenance mode, **patch NIS as necessary**. Consider making the transition to a more secure network-based administrative system. Conversely, the administrator may want to regularly white hat the NIS system using well-distributed cracker tools such as ypsnarf to ensure that the system is secure against the major exploits.

Password Security in NIS

Even if NIS administrators make their systems as secure as they can get using the above methods, they will still face the problem of password aging. It can be a critical issue for some, since their legacy systems have long had their passwords cracked. Also, administrators are likely to find that they are out of compliance with their company's password policies due to the presence of NIS systems. For instance, the federal government has an agency-level rule that passwords may endure for 180 days at the maximum. [11] Most branches of the federal government and most private corporations have stricter rules. Forced password rotation every 90 days is a common company policy. Companies running NIS must currently implement their password expirations manually or install special tools to replace yppasswd, essentially bypassing NIS and losing some of its convenience advantage.

Password aging within NIS can be implemented in a number of different ways. In order to devise the algorithm from which to generate an NIS password aging tool (here hypothetically called YP_PEX), an administrator needs to consider the efficacy of the various options as well as their practicality. In one office, it may be easy to make everyone change passwords at the same time, regardless of how recently the accounts have been created or changed. In another, this may result in a backlash against security, so it may be worth tracking account passwords individually.

In any NIS password aging system, copies of the password hashes must be stored. Security-minded administrators may have a problem with this but there is no escape. All expiration-capable password programs must make use of the old passwords in some way in order to compare them to the current passwords. Comparing hashes is a good and convenient choice for NIS. The hashes could be further encrypted or checksums could be taken against them and stored in their place but these don't seem to be good options, at least in the creation of a home-grown tool.

Determining the Password Policy

In most cases, the password policy will be determined by someone other than the NIS administrator. Policies can come from the leadership of the local office, the national level, or even the international level. Whatever the local policy, the NIS systems administrator should check to make sure it complies with the policies set at the highest level in the organization.

If the NIS administrator must develop a policy or policy implementation on his own, central points to consider are that 1) the NIS users must be educated about choosing good passwords, 2) the administrator needs to find a way to enforce good password choices, and 3) there should be a way to expire the passwords in a timely fashion. There are many articles on creating a good password policy, including some at SANS. [12, 13]

In general, there are some things an administrator would want to exclude from passwords:

- Login names
- First or last names
- Spouse's or child's names
- Repeated letters (more than two A's in a row, for instance)
- Digit-only strings (since there are less digit combinations than alphanumeric)
- Words in any standard English dictionary
- Strings shorter than 6 characters
- Other easily guessed or easily obtainable information (such as social security number or uid)
- Previously used pass phrases

An ideal, secure password will appear to be made of random characters. [14] It should mix upper and lower case, when possible, and include digits and punctuation. An NIS systems administrator creating his or her own policy should put these ideas into writing and get approval for the policy before moving to the next step, implementation.

Implementing the Password Policy (YP_PEX)

In creating a home-grown tool for expiring NIS passwords, an administrator's most likely choice for the programming language is Unix shell (sh). Many NIS tools are already written in C, sh, or csh, so such a choice is natural. It has the advantages of being easily changeable and relatively readable for other administrators. Speed of execution is not a factor for a small event that needs to take place only once per day.

The implementation issues that the administrator writing YP_PEX needs to consider are: the NIS account information to keep, comparison of current NIS password hashes to archive data, warnings sent to users when accounts are about to expire, expiration of old accounts, and possible restoration of account information. Recoverability of data must be taken into account, as should be the planning for further development as well.

Information to Keep

Out of the typical NIS password file, the main field to keep in storage is the password hash itself. The hash in this case is alphanumeric text generated by the DES cryptography used on the NIS server. There is no way to run the DES encryption backward and save to plaintext (not to mention that keeping plaintext passwords on the system would be an extremely bad idea). Fortunately, the hashes are sufficient. The same password gets the same hash from one login to another, so comparisons will work for the purposes of account expiration.

In a NIS password file, the second field demarked by the colon (:) separators is the hash, as in this example:

bd:4&51#60.7:99:Bob Doe CompName DivName Phone:/bin/csh

However, the hash by itself is not sufficient. There must be some way of comparing each hash per username (the first field above, in this case the user "bd"). For the sake of expediency (but not security) the two fields might as well be stored together. In fact, for the sake of creating a useful YP_PEX script, the NIS administrator might want to keep an archive copy of the entire password file for each day. On a DFS system, this might be considered insecure but in NIS, this generates small additional risk. The fact that the current password hashes are broadcast makes the process of archiving negligible by comparison.

It's probably best to archive the password files in a date-appended form for easy sorting. Therefore, a date command such as

```
DATE=`date +"%Y%m%d"`
```

should be used to append to the archived file, which will give a result in the form of

```
passwd.20010226
passwd.20010227
passwd.20010228
passwd.20010229
```

These can be stored in an archive directly off of the home NIS directory or anywhere else on the system. Some will feel better renaming the files or finding a more secure hiding place, perhaps even a disk or tape that is generally offline except for the daily expiration task. This is the sort of security through obscurity that many system administrators indulge in, probably to good effect. A file marked 20010228px or simply 20010228 would be less attractive to intruders with the root privileges to read it. Of course, one can argue that if an intruder gains root privilege, the battle for security on that particular machine is lost. An NIS administrator does not want to make the implementation of password archiving too difficult.

A special note on the archiving of old password files: the local password policy may permit old passwords to be reused after a year, two years, or never. Such a choice affects the length of time the archive password information is held. Some sort of removal of old password files should be done automatically as well, dependent on the policy.

Comparison of Hashes

The comparison of the hashes should take place for every entry in the password table. Each line in the NIS password map must have its first field evaluated to ascertain if it exists in both the current and archived form (that is, the account has existed for the past 90 or 180 days). If the account is found to have existed for more than the calendar threshold, then the hashes must be checked. Accounts with matching hashes should be flagged for warning, disabling, or removal depending on the password policy.

Warnings to Users

Sending a login warning to one of the startup files, which are generally .cshrc, .profile or .login depending on the users' shell, might be the best way to warn the user of the impending password expiration. However, if the user doesn't log in for weeks at a time and is somebody strategically important, perhaps the system administrator's boss, an email notification system may be called for as well. There may be a way to implement one conveniently.

In order for automated mail notification to work efficiently, some sort of email record must be kept and associated with each username. This is what some systems do anyway but it is not implemented by default in NIS. Therefore the system administrator concerned with expiring passwords and giving email notification of the tasks pending will need to do some sort of record-keeping in this regard.

As in the username/hash pair itself, an obvious place to keep such a record is the password file. (Another good place might be the distributed etc/aliases file, if one exists for the NIS server in question. For the sake of creating this hypothetical tool, YP_PEX probably should not depend on that.) A useful convention for storing a valid email address in the password file might be to keep it as the last record in the comments section. That is, in the form of:

jd:7mf932%90:99:Jane Doe CompName DivName Phone jdoe@compname.com:/bin/csh

so that such a table entry could be parsed with

grep ^jd passwd | cut –d":" –f4 | awk '{print $6}'

to get the correct email address. If the password table entries were kept consistent with one another, including dummy comment field entries where the information was unknown, the parsing statement could be kept simple. A standard warning text could be spooled out to each user in danger of having his or her account locked. Perhaps the message could even be sent twice, the first time a week ahead of the expiration and the second time two days before the account is locked.

Expiration of Accounts

Even with an adequate user warning system, NIS administrators may wish to leave themselves a way to disable logins to accounts without actually deleting them from the NIS maps. There are a number of possible methods. However, in experimenting, the administrator should *not* delete the password hash from any account. This removes the password from the account on most NIS systems, which enables users to login without one.

The simplest way to effectively lock an account is to write into the second field of the password file. Any text aside from the hash guarantees that the old password won't work. Indeed, it's doubtful that any text string at all can DES encrypt to match the contents of the edited second field.

An example of the edited second field:

jd:LOCKED7mf932%90:99:Jane Doe CompName jdoe@compname.com:/bin/csh

As a bonus, the text "LOCKED" can be easily removed from the hash by the administrator without damaging the hash or revealing the old password. This means that the administrator can re-activate any expired account in a matter of minutes if the need arises. Naturally, an NIS administrator can use some other text phrase aside from "LOCKED."

Further Development

Features such as hidden password archive directories, changed map locations, or encryption and decryption for the password files may be scripted at any time. However, it's important to remember that a password expiration tool need not be perfect to be better than the alternative of none at all. There are other password tools out there that can replace the yppasswd but few of them are cross-platform and all of them involve considerable effort to deploy. System operators who switch from NIS+ to NIS in order to gain ease of administration seem unlikely to deploy such labor-intensive tools. An archiving shell script may be a better answer, since it can be improved with revision while remaining easy to read and use.

Alternative Password Policy Tools

NIS is not monolithic. When people talk about their implementation of NIS, they are often talking about something quite different from an implementation in another operating system, let alone in another network environment. When someone says a given task can be done in NIS, it might not be possible in an older version of NIS, a newer version, or in multiplatform NIS as opposed to Linux-only or Solaris-only workshops.

One way to bridge the technology gap caused by using NIS across multiple Unix operating systems, now including MacOS X, is to use a web-based tool. There is no web freeware available for Unix password administration at this time but one can easily imagine how it could operate. Much in the fashion of the web interfaces to Microsoft Exchange, Sun Answerbook and HP Web JetAdmin, a password administration tool could validate old passwords and permit the entering of new ones. It would perform a few CGI actions in the background, keep its web traffic encrypted with SSL, and do SCP transfers of NIS map information.

Until such a tool exists, there are two established alternative Unix password programs that may allow system administrators to enforce a reasonable password policy, including expiration dates, in NIS. The extent to which they are multiplatform is debatable. The second mentioned below, npasswd, seems to be the most useful. However, it implements a hard-coded password policy that may not be acceptable to some NIS administrators.

Passwd+

When installed, the passwd+ program prompts the user for the current password and then a new one. The new password is then tested (with tests configurable from site to site [15]) to ascertain its difficulty level. Only when the chosen password meets the compliance standard of the local password policy is the user allowed to continue.

Passwd+ advertises NIS compliance but not much mention of it is made in NIS literature. A few Usenet references to it indicate there is some configuration difficulty, not insurmountable. No mention is made of the NIS password map distribution problem.

Npasswd

The npasswd documentation implies that its authors are aware that it may still give away password hash information when used with unsecured NIS. [16] Nevertheless, npasswd gets mentioned reasonably often in NIS literature. Although the documents indicate that it must be installed on all client systems and requires extensive configuration, a serious inconvenience and a major project for most administrators, it does have some advantages.

Aside from intelligent password checking, npasswd can be configured to inherit abilities from the passwd binaries it replaces. Since any multiplatform tool is necessarily limited, this functionality seems exceptionally clever. Backwards compatibility (or sideways compatibility) is preserved across all platforms for which npasswd can be compiled.

Conclusion

Even with the presence of yppasswd replacements, such as passwd+ and npasswd, there is a need for a password expiration script in NIS. Such a script could be created and updated more easily than a yppasswd replacement. It could be distributed and implemented more quickly, perhaps eventually with the default distributions on various platforms. In a cross-platform sense, it's a surprise such a script doesn't already exist. The reason may lie in the separate passwd solutions from separate vendors, including Linux.

Either npasswd or passwd+ would perform better than a mere password expiration script in many other ways. Both include intelligent password checking. Both advertise some sort of compatibility with NIS yppasswd. Unfortunately, both also apparently suffer from the same NIS map distribution problem that can be solved only by moving to a shadow password implementation that works on all NIS operating system platforms. Both may suffer from another distribution-related problem in that password strengthening generally can't be enforced except on the NIS master server. Users modifying their passwords on client systems will not be subject to the password strengthening parameters unless npasswd or passwd+ is installed there, also. [17] Nevertheless, either npasswd or passwd+ would be preferable to the standard yppasswd program.

As far as the creation and implementation of a password policy, the most important step is to start on that well-trodden road. Just as SSH1 is still useful despite its flaws (as of this writing, a new exploit has been discovered for SSH3, demonstrating that security problems in all areas continue to proliferate), a password expiration tool of some sort, however flawed, is needed in this age of well-known cracks against the NIS administration system.

References

[1] Dennis, James. "Linux Gazette: The Answer Guy, UID/GID Synchronization and Management." August 1998. URL: http://www.linuxgazette.com/issue31/tag_uidgid.html (29 June 2001)

[2] Hughes, Doug. "Securing NIS (Formerly YP)." 8 November 2000. URL http://www.eng.auburn.edu/users/doug/nis.html (17 July 2001)

[3] Craig, Jim "Re: Rootless NIS passwd Maps" 13 July 2000. URL: http://archives.neohapsis.com/archives/sf/sun/2000-q3/0083.html (2 July 2001)

[4] Bresink, Marcel. "Integrating MacOS X in an NIS Environment." 22 June 2001. URL: http://www.bresink.de/osx/nis.html (21 June 2001)

[5] Kukuk, Thorsten. "The Linux NIS(YP)/NYS/NIS+ HOWTO." 18 November 2000.

URL: http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/html_single/NIS-HOWTO.html (25 June 2001)

[6] "Using NIS." http://userpages.umbc.edu/~jack/ifsm498d/llb-nis.html (17 July 2001)

[7] Kukuk, Thorsten, "Setting Up the NIS Client." 18 November 2000. URL: http://www.opennet.ru/docs/HOWTO/NIS-HOWTO/x227.html (17 July 2001)

[8] "Psionic Portsentry, Port Scan Detection and Active Defense System." 23 July 2001. URL: http://www.psionic.com/abacus/portsentry (23 July 2001)

[9] "Patch ID# 101435-04, SunOS 4.1.3_U1: ypserv and ypxfrd fixes" 4 August 1997. URL: http://www.ibiblio.org/pub/sun-info/sun-patches/101435.readme (17 July 2001)

[10] Venema, Wietse. "README for rpcbind." 10 April 1998. URL: ftp://ftp.porcupine.org/pub/security/rpcbind_2.1.README (29 June 2001)

[11] "NIH Policy on Passwords." 20 March 2000. URL: http://wwwoirm.nih.gov/policy/passwords.html (6 June 2001)

[12] Worthington, Mark. "Creating Security Policies – Lessons Learned." 4 May 2001. URL: http://www.sans.org/infosecFAQ/policy/creating.htm (17 July 2001)

[13] Root, Carl. "Password [In]security: Common Issues Surrounding Compromised Passwords" 11 May 2001. URL: http://www.sans.org/infosecFAQ/authentic/insecurity.htm (17 July 2001)

[14] ITC, University of Virginia. "Choosing Good Passwords." 11 July 2001. URL: http://www.itc.virginia.edu/helpdesk/accounts/passwords.html (17 July 2001)

[15] Bishop, Matt. "Index Entry for Passwd+" October 1997. URL: http://www.admin.com/Pages/TUSA/passplus.html (17 July 2001)

[16] Hoover, Clyde. "Npasswd Documentation." 20 July 1998. URL: http://www.utexas.edu/cc/unix/software/npasswd/doc/ (2 July 2001)

[17] O'Brien, James L. "Security Issues in NIS." 10 November 2000. URL: http://www.sans.org/infosecFAQ/unix/NIS.htm (2 July 2001)