Proceedings of the 9th Colloquium for Information Systems Security Education
Georgia Institute of Technology
Atlanta, Georgia, 6-9, June 2005

Best Practices and Worst Assumptions

Matt Bishop

Abstract – The development of best practices and checklists to improve system security has popularized techniques and technologies for strengthening systems. These techniques provide a basis for teaching the importance of assumptions in computer and information security, and the necessity of questioning them. We present an example of analyzing a set of security guidelines to determine the underlying assumptions, and give examples of how to demonstrate the importance of the assumptions to the effectiveness of the guidelines.[††]

Index terms – Education, best practices, challenging assumptions

I. Introduction

As the discipline of computer security has matured, it has moved from an art practiced by experts to a technology being adapted for use by non-experts. The principles and practices developed by the experts are being distilled into sets of best practices and security checklists aimed at enabling people to improve the security of systems. Examples of these standards, guidelines, and checklists include the now defunct Trusted Computer Security Evaluation Criteria (also called the "Orange Book) [1], the NIST System Administration Guidance for Securing Microsoft Windows 2000 Professional System [2], and various guidelines for secure programming. All these imply that by following the development, implementation, configuration, installation, and/or maintenance guidelines, security of the resulting system is better than that of systems developed, implemented, configured, installed, and maintained without the guidelines. For simplicity, we refer to all of standards, best practices, and security checklists as "security guidelines".

Whenever one suggests that a system's configuration is "better" than another, the question of "better how" arises. The problem is that the term "security" assumes some policy defining what "security" means. Without this context, "better" is ambiguous. To measure "security" is meaningless without first stating the security policy. For example, a University workstation may be considered secure if all users can read each other's data. The purpose of a university, after all, is to share information and encourage collaboration among faculty, students, and staff. However, this system would be non-secure were it transported to a commercial environment, where disseminating information could compromise trade secrets. Similarly, in a law office, where absolute confidentiality between lawyer and client is required, such a system would be unacceptable.

When preparing a security guideline, one tailors it to a particular environment. This environment includes a security policy and assumptions about the people and procedures (including enforcement procedures) in place. Erroneous assumptions about the environment lead to security measures that are ineffective, and a set of guidelines that do not reflect reality. This in turn may lead to a false sense of security, which weakens the actual security of the system!

Part of analyzing the security of a system includes questioning whether the assumptions underlying the security measures are appropriate for the environment. An obvious example comes from the play *A Funny Thing Happened on the Way to the Forum*, in which one character decides to kill himself because of unrequited love. He is promptly told, "It's against the law to kill oneself. The penalty is death." The penalty here has the opposite of its intended deterrent effect. An example from the computer arena occurred when a major software company was going to be sued for violating an antitrust law. The company offered to supply free computers to schools as part of a settlement. The effect, of course, would be to spread the company's influence even farther — precisely what the lawsuit was trying to prevent. No one ever broke into a system by compromising a model proved secure. Attackers compromise the implementation of the model, which is *not* proved to implement the model correctly. The assumption that the system correctly implements the model is just that, an assumption — and because this assumption is often wrong, the security of the system may be compromised.

This suggests an approach to teaching the need to analyze assumptions: take the security guidelines, analyze them to see what assumptions the authors are making, and apply those guidelines to other environments where one or more of those assumptions fail. This approach has two benefits. First, it helps students understand the importance of

[†] *Author's address: Dept. of Computer Science, University of California at Davis, One Shields Ave., Davis, CA 95616-8562 USA*

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

assumptions, by showing them how guidelines proposed or recognized by some authorities fail when assumptions are changed. It also takes advantage of a quirk in human nature, namely that of challenging an authority. Second, it helps the students learn how to craft guidelines, and the importance of minimizing the number of assumptions made so their guidelines can be applied as widely as appropriate.

In the next section, we review one set of security guidelines. We then discuss some underlying assumptions. The fourth section proposes exercises to help students understand these assumptions better. We conclude with a brief discussion of the pedagogic value of this technique.

II. Guidelines

This section reviews the CIS *FreeBSD Benchmark* [3]. This is a benchmark intended for systems running the FreeBSD variant of the UNIX® operating system. The guideline was developed by the Center for Internet Security using a consensus-based process. Its purpose is to quantify how secure a system is, and to enable managers and administrators to detect when changes improve the security of the system. It is a good example of a security guideline, in part because there is an associated scoring tool that scans a system, reports on violations of the guidelines, and even computes a numerical score based on the degree of compliance with the guidelines.

The guidelines consist of 8 sections, each covering a different aspect of security (see Table 1). Within each section, the aspect is broken down into specific items. Each item has the form of an optional question designed to determine the appropriate setting, an action to be taken that configures the item, and a brief discussion of the item. Where configuration differs between FreeBSD versions, the actions for the versions from version 4.8 on are given.[1]

Section	Contents
1	Patches and Additional Software Configuration
2	Minimize *inetd* Services
3	Minimize Boot Services
4	Kernel Tuning
5	Logging
6	File/Directory Permissions/Access
7	System Access, Authentication, and Authorization
8	User Accounts and Environment

Table 1. Contents of CIS FreeBSD Benchmark

[1] FreeBSD versions 1 through 3 are outdated. FreeBSD installations use versions 4 (specifically 4.8 on) or 5.

The guidelines state certain assumptions in the introduction. The critical ones are:

1. The person testing the system, and reconfiguring it when necessary, is working as the superuser and running the FreeBSD version of the Bourne shell with the parameter *noclobber* unset. This is necessary because the actions are given in Bourne shell syntax. The script would overwrite existing files (hence the need for *noclobber* to be unset; otherwise, the overwriting would fail).

2. The actions in each item are executed in the order given. In fact, they may be copied directly from the guidelines using "cut-and-paste". If the order is changed, the operations may fail, or have an unintended result.

3. After all changes are made, the system must be rebooted. This is necessary because some actions require the kernel being rebuilt, or system daemons rereading their initialization files.

The benchmark does not provide any guidance about the environment for which it was written, nor upon the policy of the specific organization or group using the system. It does ask specific questions, and conditions the actions upon the answers to those questions. It also does not quantify assurance because it prescribes one particular setting, or set of settings, without considering the effectiveness of the mechanisms that use the setting. This is reasonable in the context of the benchmark because it assumes the same mechanisms on all systems. In contrast, more general guidelines such as the TCSEC require evidence of assurance of the mechanisms, and prescribe how the mechanism is to act rather than prescribing particular settings for a specific mechanism.

This brings us to the question of the importance of the assumptions that the guidelines make.

III. Underlying Assumptions

Inherent in all security guidelines is a particular purpose: to conform to some idea of "secure". The idea of "secure" varies from guideline to guideline. Moreover, guidelines are often not explicit about the policies they are trying to enforce. The policies, or more properly the policy components, are implicitly assumed.

This affects the applicability of the guidelines to particular environments. To take an extreme example, the policy for a system used to regulate the beating of a patient's heart in a hospital's intensive care unit would disallow access over the Internet (one hopes). So, a guideline written for that medical system would say that the system is not to be connected to the Internet. But

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

applying that guideline to an Internet-based business, such as Amazon, would result in massive economic losses and bankruptcy. The security policy of the business, in fact, must allow connecting to the Internet. The security guideline for one environment (a hospital) instantiates a breach of security in a different environment (an Internet-based business).

The scoring tool bundled with the guidelines measures compliance with the guidelines. A high rating means the system meets the guidelines, and a lower rating means the system fails to conform to the guidelines.

Assumption 0. The scoring tool reflects the guidelines accurately and completely.

This assumption is pervasive. In what follows, we interpret the guidelines in light of the actions of the scoring tool, and point out where the tool amplifies (or differs from) the guidelines.

A. Section 1: Patches and Additional Software Configuration

When flaws are discovered in the system, patches or updates are constructed to remediate the flaw. The flaw may not be related to security. The FreeBSD Benchmark says that the "operating system should be promptly patched after a security hole is located" and gives detailed instructions about how to download the patch.

The scoring tool amplifies the guidelines. It checks to see if the program *cvsup* has updated the source tree in the past month. This is different than checking that all security patches to the operating system have been installed, because no patches may have been announced in the previous month. Further, the patches may have been downloaded, but not installed. So, if a system administrator relies on the tool, the resulting message about patches may, or may not, be accurate. This leads to the first set of assumptions:

Assumption 1. Patches are released at least monthly.

Assumption 2. The system administrator uses *cvsup* to download patches.

Assumption 3. Once a patch is downloaded, it will be installed, and the relevant programs recompiled and reinstalled (and, if appropriate, the system rebooted).

The next item pertains to host authorization. When a host connects to a FreeBSD system, that system can be configured to check an authorization file to determine whether the host should be allowed to connect. The particular mechanism is called "tcp wrappers" and can be implemented by using a specific program (*tcpd*) to "wrap" servers, or by instructing *inetd* to invoke the mechanism automatically. The guideline recommends setting the "W" and "w" options, which instruct *inetd* to invoke the mechanism for all services. Interestingly, the scoring tool for FreeBSD only checks that either flag is set; this means that one can configure *inetd* to check that the mechanism is invoked for some services, rather than all services.[2] Further, if the "wrap" method is used, so that *tcpd* is explicitly invoked in the configuration file, the scoring tool asserts that this method of host authorization is not used. This leads to the next set of assumptions:

Assumption 4. Tcp wrappers is invoked using command-line options to *inetd* rather than by directly invoking *tcpd* in the *inetd* configuration file.

Assumption 5. If host authorization is provided for either internal or external servers, then it will be provided for both internal and external servers.

Although this item states that its goal is to "enable [tcp wrappers] with *inetd*,"[3] it does not provide information on other servers that use this mechanism.

Assumption 6. All network services that require host authorization are either started by *inetd* or are explicitly described in the benchmark (*sshd*).

The third item recommends enabling *sshd*. It presents a small script to enable version 2 of the protocol, allow *root* logins, and the printing of the message of the day, "/etc/motd". The first assumption here is that the site requires remote access to the system. If remote access is *never* authorized, then there is no reason to run *sshd* (and in fact it should not be run). Hence:

Assumption 7. Users will need to access the system from a network.

This also assumes that version 1 of the protocol should be disabled. But if some users authorized to use *ssh* will have access to version 1 clients only, this causes problems:

Assumption 8. Users authorized to use *ssh* will never need to use version 1 clients.

The second assumption is that remote *root* logins are not permitted. In general, allowing them is undesirable, because of the need for accountability. In FreeBSD, *root* is a role account [4], and it is typically shared among many users. (See section 3C below.) If a *root* user is

[2] Specifically, "w" refers to *internal servers* that *inetd* handles without invoking another server, and "W" refers to *external services* (such as FTP) that *inetd* handles by invoking another server (such as *ftpd*).
[3] [3], p. 3.

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

logged in, who is the actual user? When users use *su* or *sudo* to acquire *root* privileges, the action is logged. But a remote login as *root* records that *root* logged in, not who was logging in as *root*. This eliminates the ability to tie that instantiation of *root* back to an individual.

Assumption 9. Remote logins as *root* are unnecessary.

The final assumption is a tricky one, and unintended by the authors of the standard. There is a typographical error in the action because a quotation mark is left off one entity.[4] If one knows the program *awk*'s command language, the error is obvious; but if not, the user may be puzzled. Hence:

Assumption 10. The system administrator is familiar with *awk*'s command language.

B. Section 6: File/Directory Permissions/Access

Section 6 deals with the permission modes of some system and user file objects. The first item considers the user and group databases. In FreeBSD, password information is distributed among 4 files. The first, "/etc/passwd", contains the information that is to be accessible to users. The second, "/etc/master.passwd", contains other information such as aging and password hashes, and should not be generally readable. Each of these files has a binary version for quick access. Finally, group information is kept in the file "/etc/group", which needs to be readable by everyone.

The guideline recommends changing the permission of "/etc/passwd", the corresponding database, and the group file to allow the owner to read and write, and the group and world to read only; and to change the permission of "/etc/master.passwd" and its corresponding database file so that only the owner can read and write it. The implicit assumption here is that only *root* will need to alter any of the files, and read the master password file and database.

Assumption 11. Only *root* needs to alter the password and group files, and read or alter the master password files.

Examining the scoring tool, two other assumptions become apparent. The guidelines recommend changing the group of these files to *wheel*, but the tool tests that the group owner of the file has GID 0. Similarly, it checks the ownership of the files against the user identification number (UID) 0, even though the guidelines recommend changing the owner of these files to *root*.

Assumption 12. The UID of user *root* is 0, and the GID of group *wheel* is 0.

The second item states that user home directories should be kept private by making them accessible only to the owner. This action makes subdirectories and files in the directories inaccessible to other users. It restricts sharing, in the sense that the users must take special action to make data available to one another without violating the guideline (sending documents via email, for example), and will break other services such as web page serving, where users keep their web pages in a subdirectory "public_html".

Assumption 13. Data in user directories never needs to be shared. This includes web pages.

The discussion contains a warning that users should be warned before this change is made, and that the change should be made with caution. As an alternative, it suggests turning off world read permission on home directories, but setting owner and group read, write, and search permission on that directory. This recommendation assumes that group writing to home directories is acceptable, and home directories are not world-writable (because the write permission for world is not cleared). Hence:

Assumption 13½. Owners wish to allow group access (read, write, search) to their hone directories, and do not have permissions set that allow anyone else to write to the directory.

The scoring for the second item checks that all world permissions are turned off, and that the group write permission is disabled if the group of the owner contains more than one user. Thus, the scoring does not reflect the guidelines, in that if group read and search permissions are turned on, the scoring tool will accept the directory as meeting the guidelines. This leads to another assumption:

Assumption 14. Group read and search permission are not relevant to the security of the system.

C. Section 8: User Accounts and Environment

Section 8 presents recommended settings for various aspects of user accounts and environments. The first recommendation is to block login access to system accounts. The action is to set the *uucp* account's shell to an invalid shell (thereby preventing logins). The discussion warns that if the *uucp* account is to be used, this action should not be taken. The scoring tool adds more information. A "system account" is defined as one with UID between 1 and 499 inclusive. Thus, the underlying assumption is:

[4] Line 3 ([3], p. 3) in the *awk* script reads:
/^#PermitRootLogin/ { $1 = "PermitRootLogin};
and is missing a closing quotation mark after PermitRootLogin.

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

Assumption 15. No user accounts have UIDs in the range 1 to 499 inclusive.

The next item is to check that there are no accounts without passwords. The assumption here is that there are no utility accounts, for example a "date" account that prints a date and exits immediately.

Assumption 16. Only users with passwords may access accounts through login.

The third item requires that password aging be enabled, and sets a 91 day expiration period for each password. Only accounts with passwords should have aging enabled; accounts with password authentication blocked should not have passwords expired. The scoring tool defines blocked passwords as those with the following hashes: "LK", "*LK*", "*", "np", or "!!". This leads to the first assumption:

Assumption 17. Accounts with password authentication blocked will be blocked using a program, rather than by direct editing of the password file.

A much larger assumption is that password aging enhances security.

Assumption 18. A system that uses password aging is more secure than a system that does not.

The fourth item changes the account creation template to reflect the password expiration times. Unlike the check in item 3, the check here expects the expiry time to be in months, weeks, or days; if hours or some other form is given, the scoring tool rejects it. Hence:

Assumption 19. The password expiration time in the user account template is given in months, days, or weeks.

The fifth and sixth items look for accounts with UID 0 (that is, for *root* accounts other than the one named *root*). The fifth checks for the account *toor*, which is typically a *root* account with the shell set to the Bourne shell rather than C-Shell (which the account *root* uses).

Assumption 20. If an account named *toor* is present, it has a UID of 0.

The sixth recommends deletion of all accounts with a UID of 0. The scoring tool looks for UIDs of 0 here, and if the account name is anything other than *root*, it reports the problem. The assumption here is that there should only be one account with a UID of 0.

Assumption 21. A system with multiple accounts with UIDs of 0 is less secure than a system with only one such account.

D. Summary

To organize these assumptions, consider a site's security policy. It contains several components. For our purposes, we consider the following components:

- Network: these control what services are to be provided over the network, and how remote authorization of hosts is to be handled. It does *not* control remote user access. Assumptions 2, 4, 5, and 6 speak to this.
- User account: these control the setting of file and directory permissions, and management of those accounts, including account names and privileges. Assumptions 13 (13½), 14, and 15 speak to this.
- System access: this controls how users log into the system, including remote user access, password expirations, and authentication. Assumptions 7, 8, 9, 16, 17, 18, and 19 speak to this.
- System account: these control the setting of file and directory permissions for the system, and management of system accounts, including account names and privileges. Assumptions 11, 12, 15, 20, and 21 speak to this.
- System administrator: these control the requirements and actions of system administrators, including subscribing to mailing lists and the rules for patching the system. Assumptions 1, 3, and 10 speak to this.

Now let us consider what these assumptions tell us about each set of policy components.

IV. Analysis of Assumptions

The goal of the student exercises is to show how the guidelines improve security for the set of assumptions that the guidelines make. We approach this backwards. We ask whether a system can follow the guidelines yet be obviously non-secure, or fail to meet the guidelines and be obviously as secure as one that does.

This approach provides several benefits. First, by following the guidelines, we must alter the assumptions in order to produce a non-secure system. Second, this is a

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

standard technique for understanding the weaknesses in a system: what assumptions did the designers and implementers make? Attackers will determine one or more of these assumptions, and then try to make them untenable to open a security hole. Third, students enjoy finding ways to dodge or evade rules, so by positing the guidelines as "rules," the teacher can play upon this rebellious desire to guide the students towards a deeper understanding of security. After all, learning should be fun!

We introduce three types of errors. A *false positive* (or *type I*) error occurs when a system follows the guidelines but is not secure. A *false negative* (or *type II*) error occurs when a system does not follow the guidelines but is secure. Keeping Assumption 0 in mind, we assume that the scoring tool reflects the correct interpretation of the guidelines. If it reports a system does not conform to the guidelines, we treat that as a report of a security problem; if it reports a system conforms to the guidelines, we treat that as a denial that there is a security problem.

A. Network Policy Component Assumptions

Assumption 2 assumes the system is connected to a network and that the host can contact one of the FreeBSD *cvs* servers. This means that the site allows outgoing network connections for remote sites with the appropriate port numbers (2401 and 5999, among others). If these ports are blocked, the *cvsup* execution will fail, resultng in a false positive because the system administrator cannot use *cvsup* to download patches.

Now, consider assumption 4. Suppose the system administrator configured *inetd* to invoke *tcpd*, the tcp wrapper program, from the configuration file. In this case, the option "W" is unnecessary. If all internal services are disabled, so the "w" option is not given either, the scoring tool will report that the system fails to satisfy element 1.2 of the guidelines. This leads to a false negative.

Assumption 5 shows the converse of assumption 4. If the "w" option is given, and external services are provided without explicitly wrapping them with tcp wrappers in the *inetd* configuration file, the scoring tool will report that the system satisfies guideline 1.2 (as only one of "w" or "W" is checked); yet this system is less secure than one in which the external servers are wrapped—a false positive.

Finally, assumption 6 points out that servers started at boot time and that are neither started by *inetd* nor explicitly in the guidelines need not have host authorization. This is a dangerous assumption, because the need for host authorization should depend upon the *type* of service and not how it is started. For example, FTP servers are normally started by *inetd*, but need not be. Similarly, web servers are normally *not* started by *inetd*. Yet failing to wrap these services may allow connections from hosts that the system administrator wants to refuse access to. A system following the guidelines would, under the inverse of this assumption, cause a false positive. In other words, the guideline's recommendation is too narrow.

B. User Account Policy Component Assumptions

For assumption 13, consider a system in which the users must share data among themselves. Guideline 6.2 requires that the user directories be inaccessible by anyone other than the owner. Thus, if assumption 13 is false and the policy allows sharing, the guidelines lead to a denial of service, which is a false positive. The guidelines themselves note that the recommendation may not always be easy to follow, and suggest that a "more realistic approach would be to use *chmod* with the *u=rwx,g=rwx,o–r* flags." The problem with this recommendation is its goal is to make the home directory searchable *only* by other users, as assumption 13½ states. Instead, it denies other users the ability to list the contents of the directory ("*o–r*") unless they are in the directory's group (*g=rwx*). Hence, if this alternative is used, the home directory could be world writable (so anyone could add or delete files) and world searchable (so anyone could access files that were known to be in the directory), and the system would satisfy the guideline (false positive). Note that the scoring tool does not accept systems configured with the alternative settings, instead reporting them not to meet guideline 6.2.

The scoring tool checks that all world permissions are turned off, and that the group write permission is disabled if the group of the owner contains more than one user. Assumption 14 points out the effect of this implementation. Thus, a system with group read and search permissions turned on for home directories will pass this check whether or not the group contains only the owner of the file. Hence, a system that is reported to meet the guidelines in fact does not, meaning a system that is not secure according to the guidelines is reported as secure. But the denial of service comment for the previous assumption applies here, even more so because group access is commonly provided to enable collaborators to share files—and that requires read, write, and search access to the relevant directories.

Assumption 15 is that any account with a UID under 500 is a system account. It is traditional to use low integers as system account UIDs, but the limits of the range have never been set. On a distributed FreeBSD system, the system account with the highest UID is "www", with UID 80. The *adduser* program, which is used to add users to the system, would therefore begin adding user accounts with UID 81. This leads to a false negative.

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

C. System Access Policy Component Assumptions

The first three assumptions in this group refer to guideline 1.3, enabling *ssh*. If a system is not to be accessed from the network (negating assumption 7), then *ssh* need not be run. The Principle of Economy of Mechanism [5] says that it should not be run. Hence a system following the guidelines is less secure than one violating the guidelines—a false positive.[5]

Assumption 8 concerns a potential denial of service. If a user is traveling and wishes to log in remotely, he or she must use *ssh*. However, if the system on which the client resides only has an *ssh* version 1 client, the user cannot access the system unless the keys are set up appropriately (version 2 *ssh* supports algorithms other than RSA). Thus, unless the users only use *ssh* version 2 clients remotely, they may be blocked from logging in remotely and performing necessary tasks, creating a false positive.

Assumption 9 follows from the guideline that *root* should never be allowed to log in remotely. But suppose a program like the backup or dump program needs to be run automatically. This program typically requires *root* access to the system. Thus there are two alternatives. The first is to create a *setuid* program, or *root* server, that is accessible over the network. The first means that another *setuid* program is present on the system to gather the data, and make it available to a non-privileged backup user who will download the data over the network. This data is therefore visible to that user locally. The second requires the server has protection sufficient so that only authorized clients may access it. This is the type of protection that the *ssh* service provides. Hence, in this case, it is prudent to allow *root* to log in remotely to perform the dump. Such a system is non-secure according to the guidelines, but not allowing this instance may make the system backups unavailable, creating a worse security hole under some policies—a false negative.

Assumption 16 is that *uucp* and other system accounts never need to be logged into directly. As the warning in the guidelines points out, this may not be true for *uucp*. This implies that an account such as "date", which prints the date and immediately exits, does not exist. For a public access system, such an account may provide useful information without compromising security. The scoring tool report is a false negative.

Assumption 17 deals with blocked accounts. "Blocked" refers to accounts that exist and are useable, but cannot have passwords authenticated (and any attempt to do so will fail). These accounts are used for system functions, such as owning system files. The scoring tool assumes that blocked accounts have the password field of the "/etc/passwd" file set to one of several values. A popular way to disable password authentication is to place an "*" or an "!" *before* the existing hash. This prevents the password from being hashed to the stored hash value, yet allows the original password to be restored easily (just delete the "*" or "!"). The scoring tool will consider these blocked passwords as valid, and complain if they are not set to expire—a false negative.

The truth of assumption 18 is debatable. The goal of password aging is to force a user to change his or he password before an attacker is expected to guess it. There is little empirical evidence that forcing users to do so prevents this. Techniques to enforce aging either rely on keeping the last *n* passwords selected, or on restricting password changing to a particular time interval of *m* time units from the last change (before which changes cannot be made) to *k* time units from the last change (after which a change must be made). The former leaves information an attacker could use to help guess passwords. The latter may block a necessary password change (for example, if an account is compromised before *m* time units since the last change). Thus, a system that does not enforce password aging may be as secure as one that does, in which case this is a false negative.

The scoring program assumes that the password expiry time will be given in months, days, or weeks, as Assumption 19 states. But the manual page for *adduser.conf*, the configuration file for adding users, allows the password expiry time to be given as a date, or in minutes, hours, or years, also. Settings of minutes and hours can be used to force a user to change the account password soon after account creation. If one accepts password aging as a useful security measure, the flexibility provided by these options is also useful, leading to false negatives if those options are used.

D. System Account Policy Component Settings

Assumption 11 asserts that only *root* is authorized to edit the password files directly. Assume not. In order to prevent unauthorized users from doing so, either the files would need to be owned by a non-*root* user (which would create many problems) or group permissions must be set to allow alteration, and users authorized to alter the files must be put into those groups. An attacker who gained access to that group, or the account of any member of that group, could gain *root* access—a bad idea. Hence this assumption cannot be altered without changing the operating system. This is also true for assumption 12, as changing the superuser's UID to something other than 0 would require the kernel to be changed, and changing the *wheel* group's GID to something other than 0 would require programs such as *su* to be changed (because they check for group membership).

[5] Vulnerabilities in some versions of *ssh* are exploitable locally [6]. Otherwise, this point would be theoretical.

Assumption 20 is based upon *toor*'s being an alternate form of the *root* account. But the *root* privileges require that the UID be 0, and the tool does not check for this. As "Toor" could be a proper last name, this account could exist legitimately without a UID of 0. This would cause a false negative.

Assumptions 20 and 21 are based upon the belief that multiple *root* accounts make a system weaker than one such account because an attacker need compromise only 1 out of *n* accounts, rather than exactly 1 account. But this theory does not take the password management problem into account. In order to ease the problem of distributing the *root* password, many users are given accounts with UIDs of 0, effectively granting them superuser privileges. There is no evidence that either approach produces a more secure system; in fact, the evidence suggests that procedural issues, such as determining who should have access to the superuser privileges, have a far greater effect on the security of the system.

E. System Administrator Policy Component Assumptions

Assumption 1 may be false. In 2004, for example, no patches were issued between June 30 and September 19, or October 4 and November 18. Hence, from July 31 through September 18, and from November 5 through November 17, the scoring tool would have reported that the system failed to meet guideline 1.1, when in reality it had—a false negative.

Once the patch was downloaded and the relevant sources updated, the guidelines assume that the patched sources will be rebuilt and reinstalled, and the system rebooted if necessary. A system administrator may fail to do this, yet the scoring tool will fail to detect this—a false positive.

Finally, if the system administrator does not know *awk*'s command language, he or she will know that an error has occurred during the action associated with guideline 1.3, but not be able to repair it. Then the system administrator must figure out how to make the changes on his or her own—and that may lead to them simply not being done.

V. Conclusion

This paper has demonstrated how to take a set of security guidelines, look at the underlying assumptions, and ask what happens when those assumptions are incorrect.

This technique is effective for encouraging students to challenge guidelines and security checklists. The problem with these items is not that they are bad. Far from it; they encourage people to comply with minimal security requirements. The problem with guidelines and lists is that they are absolutes, and the users and managers demanding conformity to those items must accept deviations when the security requirements, and policy, of the site warrant it.

The exercise proposed above will help students see beneath the security recommendations, and help them determine *why* those recommendations are being made and what those recommendations *assume* about site policy and capabilities. This in turn emphasizes to them the fragile nature of security, and will deepen their understanding of the role of assumptions in security, and in a lack of security.

Acknowledgement: Thanks to Sophie Engle for useful comments and discussions. This work was supported by award CCR-0311723 from the National Science Foundation to the University of California at Davis.

VI. References

[1] Department of Defense, *Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD (Dec. 1985).

[2] Souppaya, M., Harris, A., McLarnon, M., and Selimis, N., *System Administration Guidance for Securing Microsoft Windows 2000 Professional System: Recommendations of the National Institute of Standards and Technology*, Special Publication 800–43 (Nov. 2002).

[3] Center for Internet Security, *FreeBSD Benchmark v1.0.4 (FreeBSD 4.8 and Above)* (2003); available at http://www.cisecurity.org.

[4] M. Bishop, "Collaboration Using Roles," *Software—Practice and Experience* **20** (5) pp. 485–497 (May 1990).

[5] J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE* **63** (9) pp.1278–1308 (Sep. 1975).

[6] CERT, *Multiple Vulnerabilities in SSH Implementations*, Advisory CA-2002-36 (Dec. 2002); available at http://www.cert.org/advisories/CA-2002-36.html.