507.5
Advanced Unix Auditing and Monitoring

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

Copyright © 2016, The SANS Institute. All rights reserved. The entire contents of this publication are the property of the SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND THE SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, the SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by the SANS Institute to the User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between The SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO THE SANS INSTITUTE, AND THAT THE SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND), SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to the SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of the SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of the SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

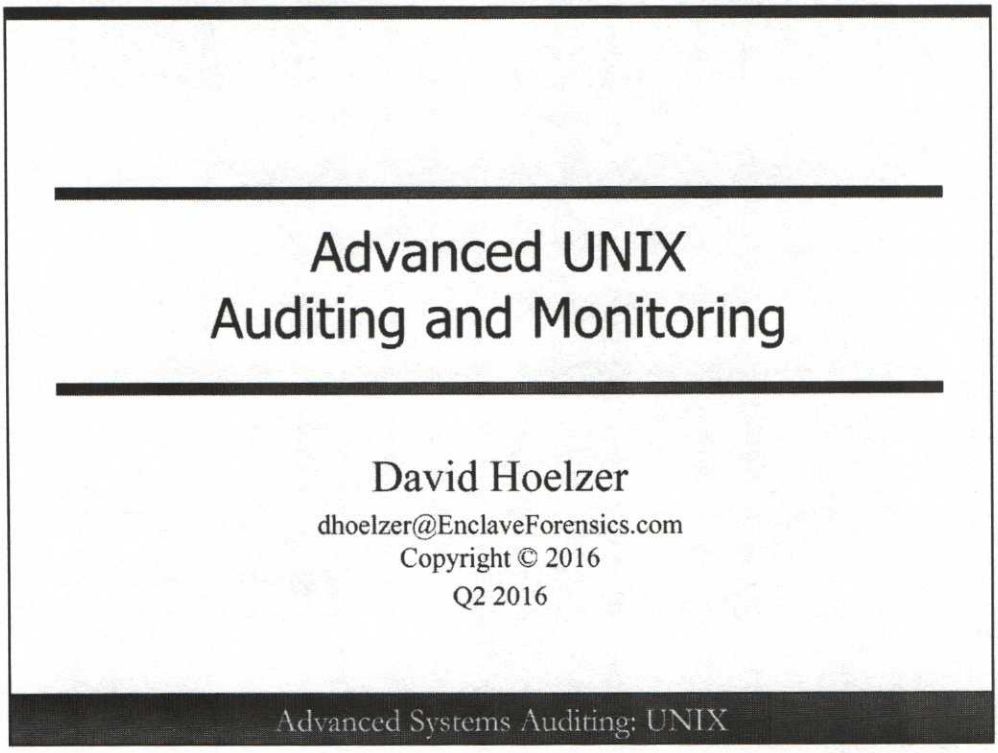
Copyright © 2001-2016, All Rights Reserved, David Hoelzer & Enclave Forensics™.

All best faith efforts have been made to properly credit any material referenced herein. If you discover any material that has not been properly referenced, we welcome your comments and corrections.

Reproduction of any kind is prohibited without express written consent of the copyright owner. The SANS Institute™ is granted license to reproduce and distribute this book in connection with authorized SANS training. Please see the SANS Courseware License Agreement (CLA) for more information regarding your rights as a purchaser.

ISBN 978-1-937060-06-0
Tenth Edition

This page intentionally left blank.

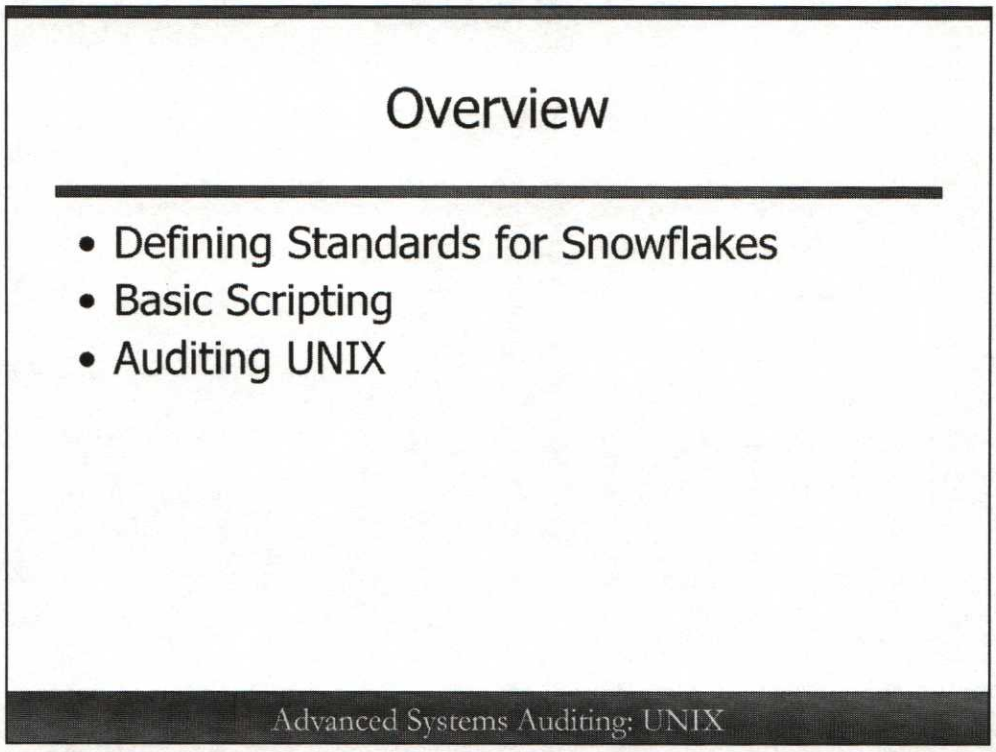


UNIX System Auditing

I always invite comments and feedback on my courses. Please feel free to write if you feel that I have omitted something important, you have some other constructive criticism, or you come across some really cool tools that would make my life easier. I ask only that you put "SANS" somewhere in the subject of your message, or it might fall prey to my spam filter!

dhoelzer@EnclaveForensics.com

© 2016 David Hoelzer & Enclave Forensics™



During the course of today's material, we will cover everything from basic UNIX concepts and configuration clear through to advanced auditing techniques, stopping off at security configuration, maintenance, logging and a how-to for a fairly comprehensive baseline audit program. We do expect that you have a basic working knowledge of UNIX coming into this course, just as we did with Windows. If you're not sure that you're up to snuff, you will definitely want to look at the appendix at the **back of this book** and work through the first exercise or two in the workbook to improve your familiarity.

© 2016 David Hoelzer & Enclave Forensics™

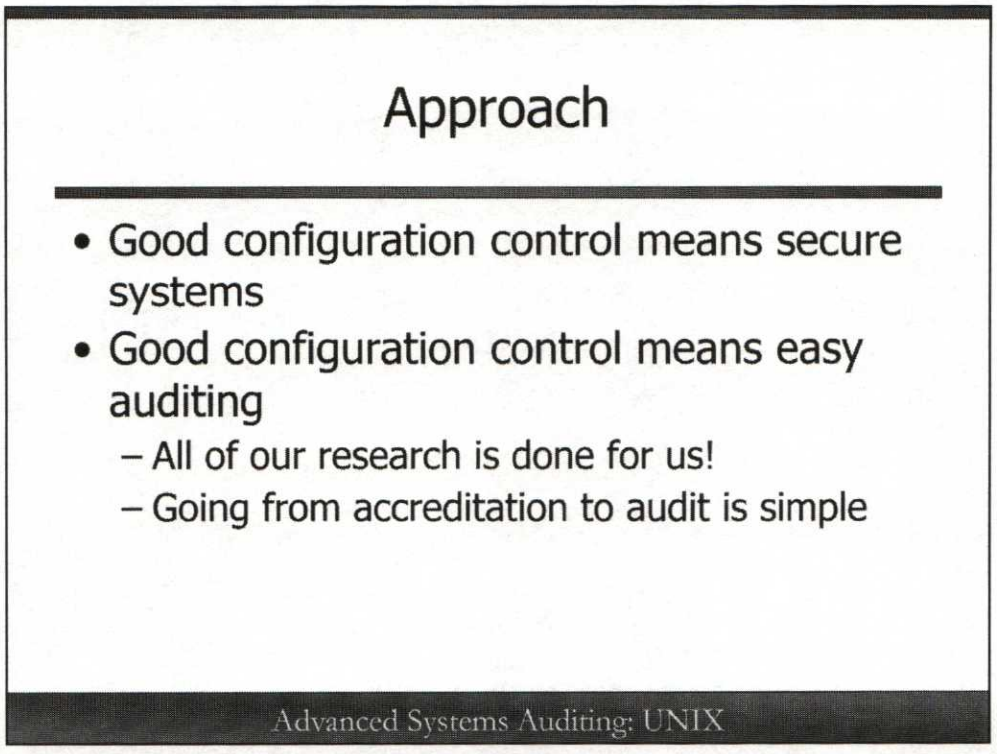
Roadmap

- Accreditation
  - Sources
  - Using
  - Creating checklists

Advanced Systems Auditing: UNIX

Our first module on our path to auditing UNIX systems is "Accreditation to Audit." The idea is to take a formal system accreditation process and turn it into our audit process!

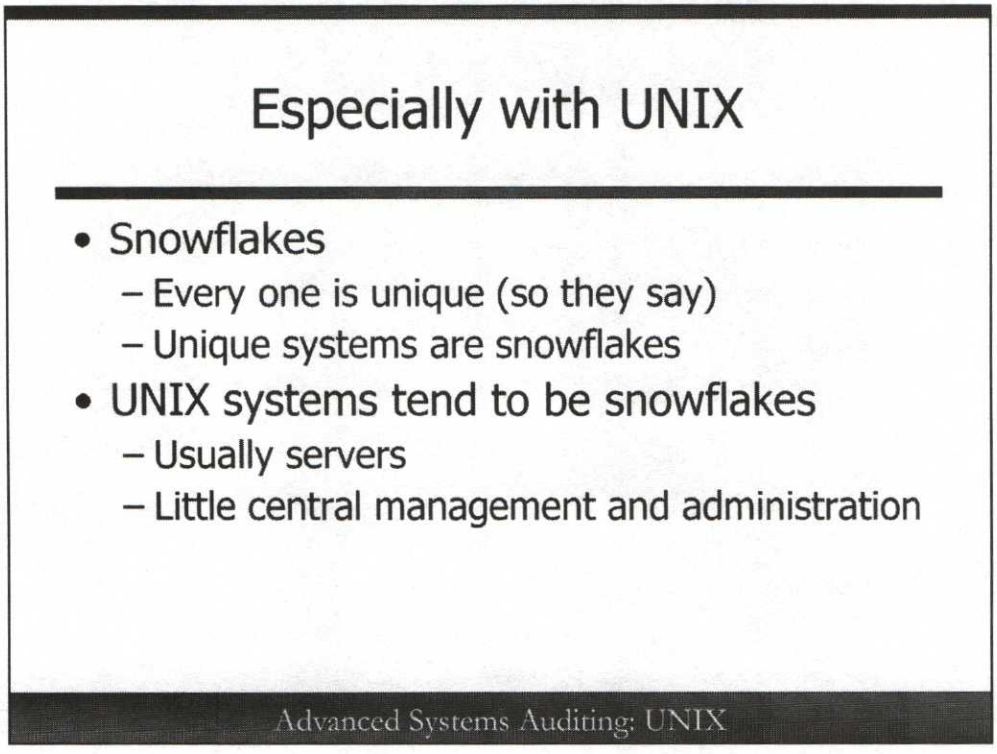
© 2016 David Hoelzer & Enclave Forensics™



The idea behind an accreditation process is that there is some strong form of configuration control implemented within the organization. This configuration control could be to the point where every system is built using a default image, and then whichever individual options need to be turned on for the system to perform its function are configured. When it comes to UNIX systems, this is actually far more work than it is under operating systems like Windows.

With that it mind, it might be better to apply a security accreditation process whereby we require that a somewhat formal risk assessment of the system is performed according to a set of guidelines that have been laid out by the organization. The system administrator, in the process of completing the accreditation process, fills out an accreditation form that includes this risk assessment. The resulting form becomes the basis for our audit of that system because, if the system is permitted to be brought into production after the accreditation process, then the accreditation forms must contain all of the relevant information about that system from an information flow perspective!

© 2016 David Hoelzer & Enclave Forensics™

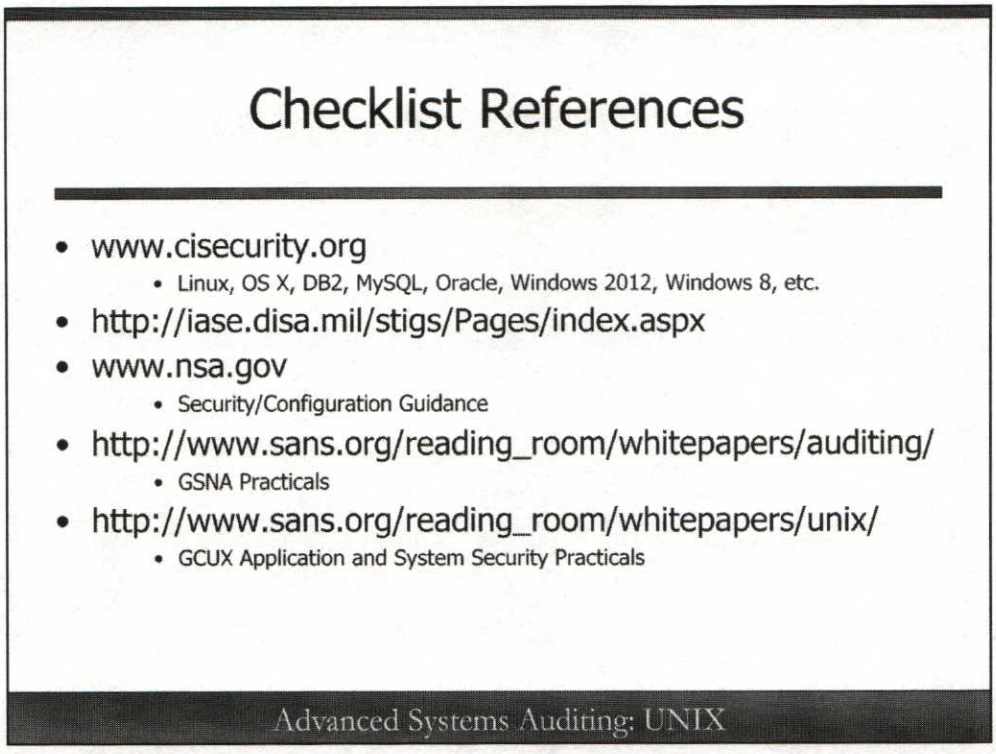


This information regarding accreditation is of equal value for all kinds of systems, but we chose to talk about it in the UNIX material rather than the Windows material because our UNIX systems tend to be less standardized than our Windows desktops. The term that's used to describe systems that vary from the norm within an environment is "snowflakes." Likely, you can derive the meaning, knowing that it is said that every snowflake is unique. Although a few snowflakes are quite beautiful, you might also know, depending upon where in the world you come from, that when a bunch of snowflakes gang up on you, you're in for a bad day.

The same is true in our networks. One or two snowflakes, no problem. What happens when all of the systems are snowflakes, though? That's a real nightmare! UNIX systems tend to be of this variety because these systems are usually deployed as servers and each server has a different role. This means that they will inherently be different. UNIX system administrators also tend to feel a greater sense of control over their systems, perhaps even a territorial inclination, which tends to mean that these systems will become more and more customized over time.

This isn't necessarily a bad thing, but it does mean that we need to think seriously about how we can manage these systems, especially from the point of view of auditing. The concept of using an accreditation system is exactly what we need.

© 2016 David Hoelzer & Enclave Forensics™



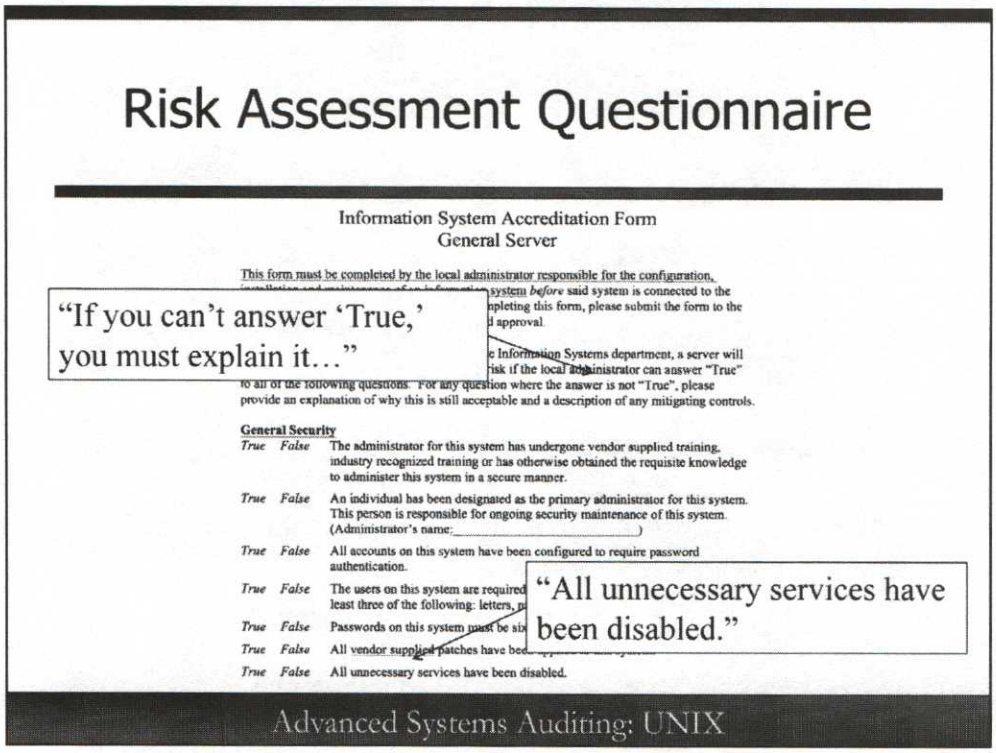
If your organization already has a well-defined standard for the deployment of secure UNIX systems, that would be the starting point for the development of an accreditation worksheet. If you do not currently have well-defined standards, someone, likely the security officer, should use an example such as those on the slide to derive a list of items that are requirements for security within your organization. You'd prefer that this list be abstracted from a specific operating system brand so that it can be applied to any operating system or at least a class of operating systems.

The Center for Internet Security has begun releasing benchmarks and tools to allow you to audit systems as compared to the consensus benchmark for due diligence.

DISA has started a repository of security review requirements documents and checklists for all DoD and government sites that is pretty useful and updated fairly frequently. This is definitely a site worth looking at for an accreditation source.

You will also want to check out many of the practical assignments written as practicals for the UNIX Track Certification. There are checklists there for most flavors of UNIX and many UNIX applications.

© 2016 David Hoelzer & Enclave Forensics™



An example of what this type of accreditation or risk assessment worksheets might look like is included on the USB Stick. It is entitled, "Information System Accreditation Form."

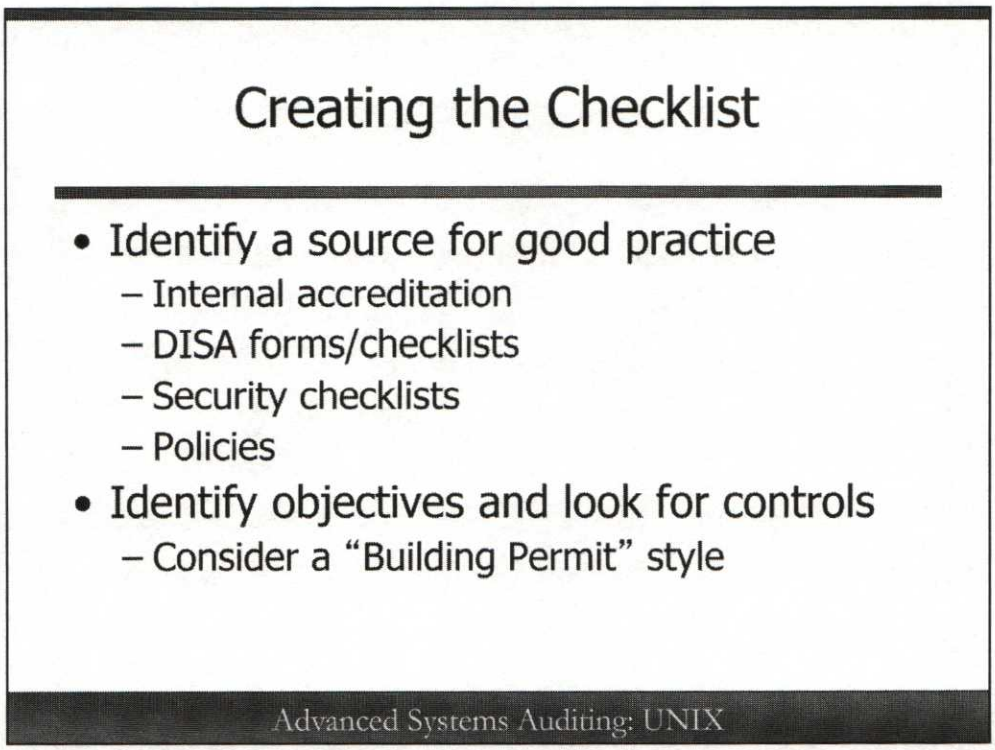
Consider the short selection of questions that we've included on the slide. Notice the last two, for example. True or false, all vendor-supplied patches have been applied to the system. True or false—all unnecessary services have been disabled. The worksheet is generic enough to be applied to any system that you might have in your enterprise yet simultaneously provides the administrator with what amounts to instructions on the proper configuration, which will allow him to get the system connected!

If the administrator answers any question "False," he must then explain how that issue has been otherwise remediated. The completed form is then submitted to the security department for review by someone authorized by the security officer or by the security officer himself. Unless there has been some previous experience with this administrator that would require a spot check and provided there are no unusual issues in the questionnaire itself, the person evaluating the risk assessment worksheet can issue a provisional approval. Of course, these individuals may perform spot checks whenever they deem it necessary, much as a building inspector would tell you to call him for an inspection before you put sheetrock up, allowing him to see the electrical work.

Finally, when it is time for us to perform an audit, ultimately accrediting the system, we simply pull the customized risk assessment worksheet that *has already been approved!* In other words, rather than having to create a customized checklist for this system, we can use the approved sheet that defines acceptable risk for that system. Again, like a building inspector, I am no longer concerned with what the letter of the law is. I now simply verify that the administrator has followed the approved "building plans" with approved "variances" from the standard requirements!

Of course, you can always provide observations and other feedback to the administrators and management, allowing the entire process to become refined over time.

© 2016 David Hoelzer & Enclave Forensics™



So, then, creating a checklist to audit a system turns out to be a pretty simple task once you have a good idea of how that system operates. This is again why the "Research" phase of an audit is probably the most critical! For our sources, we have identified several possible organizations that could assist with free information, in addition to our own organizational documentation and policies.

After we decide which sources we will use for good practice for an audit, we now need to identify various objectives, and then find controls that are in place to meet the objectives. Simultaneously, we are identifying audit controls that allow us to monitor or check on how the security controls are functioning.

One thing that I've learned works really well is to create something analogous to a "Building Permit" and "Zoning" system. Here's what I mean. To add a detached garage to your property, you would need to go through the local building department. If your property isn't zoned for the type of building that you want to do, you need to apply for a variance. After that's done, you have to submit building plans that demonstrate that you will follow all of the applicable building codes. Does an inspector need to come and check every nail and screw? No. The inspector comes only at major milestones and spot checks your progress. He uses your approved plans to determine whether or not you are still "legal."

This, it turns out, is a fantastic way to manage compliance and to make auditing of specialized systems, which UNIX systems tend to be, very easy! The administrator must complete a Risk Self-Assessment form and submit it to the security team. For anything that he answers in a way that indicates risk, he must also explain what compensating controls have been put in place. If everything seems reasonable, the security team approves his "blueprints." Now when the auditor needs to check compliance, he can get a copy of the checklist that the administrator filled out and was approved. The system when checked *must* match the sheet that was submitted!

© 2016 David Hoelzer & Enclave Forensics™

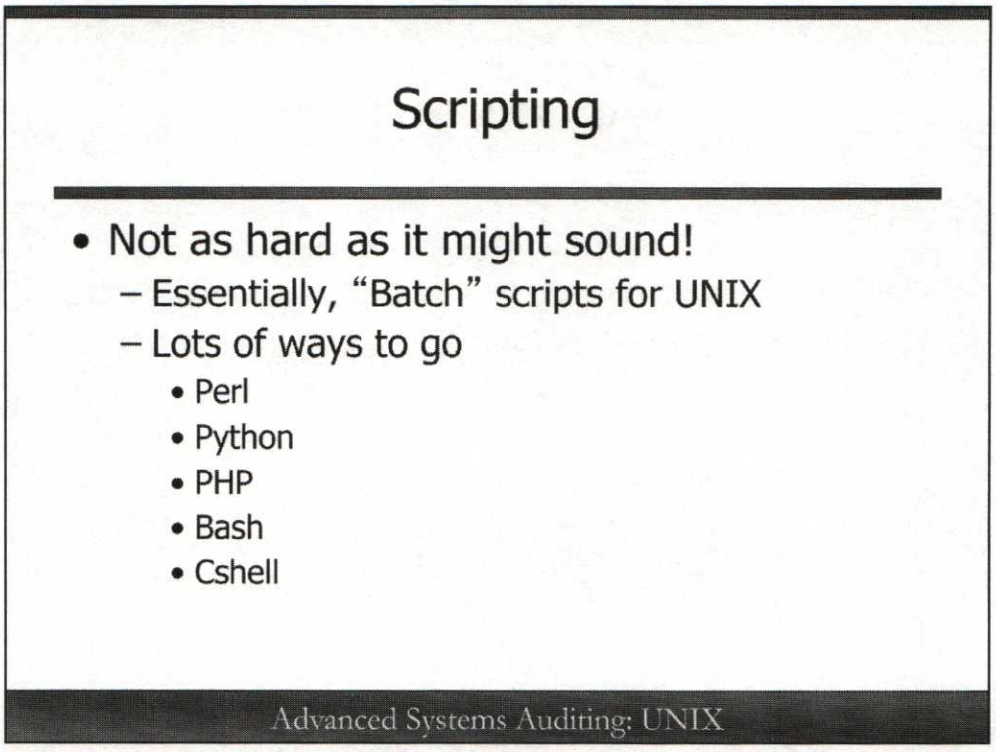
Roadmap

- Scripting
  - What's Available
  - How To Do It!

Advanced Systems Auditing: UNIX

Our next module on our path to auditing UNIX systems will bring us into the wonderful world of scripting. In this section, we cover the basics of UNIX scripting and use this as a foundation for the remainder of the day, building a fairly good set of audit scripts over the course of the day.

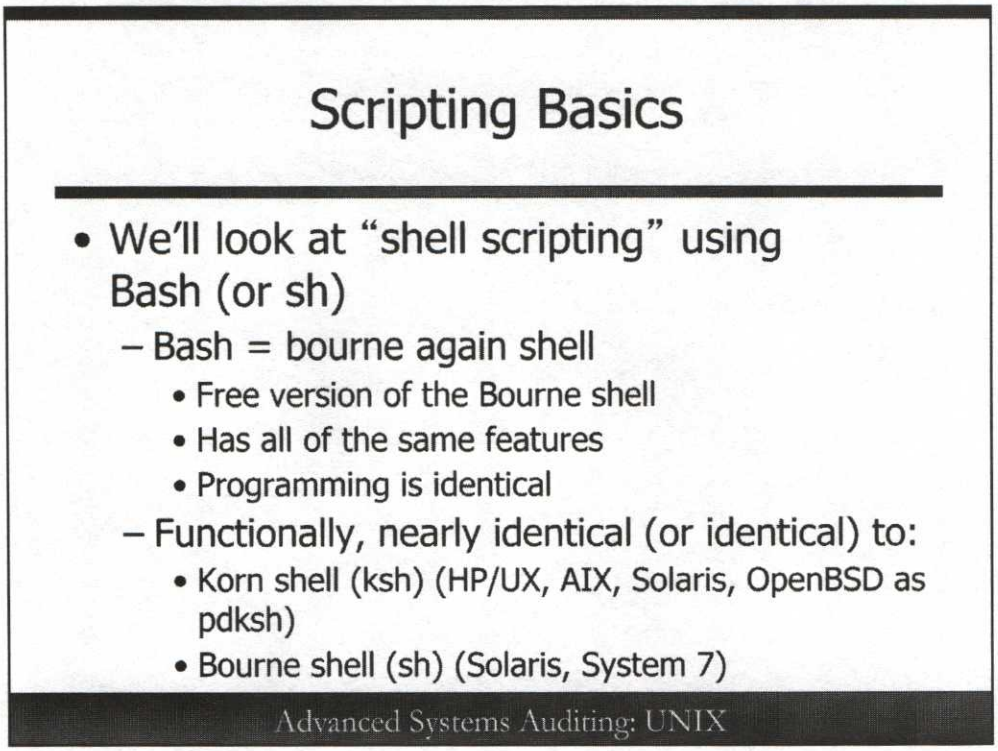
© 2016 David Hoelzer & Enclave Forensics™



Another thing that we can do to simplify our auditing tasks is to write scripts! The scripts that we write can be used in conjunction with Cron to create regular, automated audit reports. They can also be used to create scripts that we can run when trying to collect information from a system.

Scripting might sound intimidating to you if you've never done any, but don't be afraid. Knowing just a few scripting basics will allow us to create some very effective scripts. There are lots of different scripting languages available to you if you are going to audit UNIX machines. For our purposes, we will look at some very simple bash scripting because bash, or something very much like it, can be found on virtually any UNIX machine.

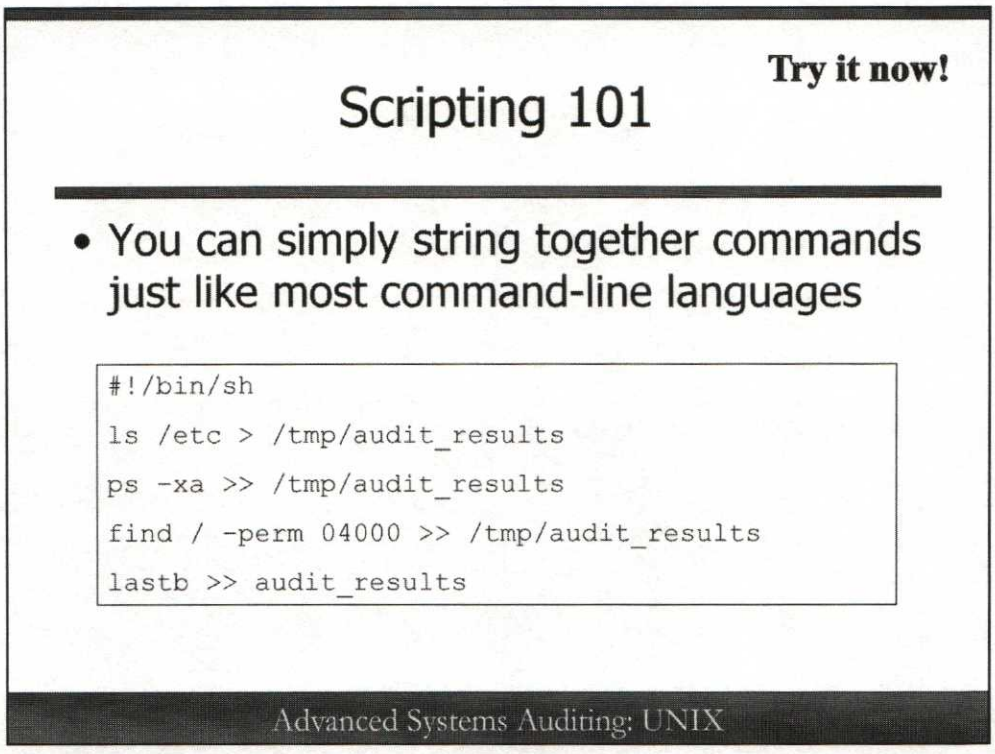
© 2016 David Hoelzer & Enclave Forensics™



The bash shell, or "bourne again shell," is a free (GPL) version of the Bourne shell. Many of the shells that you will use on a UNIX system support all of the basics that we will cover this afternoon. This means that you can take everything that we talk about here and any scripts that we write and use them on most other UNIX-based systems with little or no modification.

Even if the features or syntax are slightly different, you can always use the online manual (man) to look up the details.

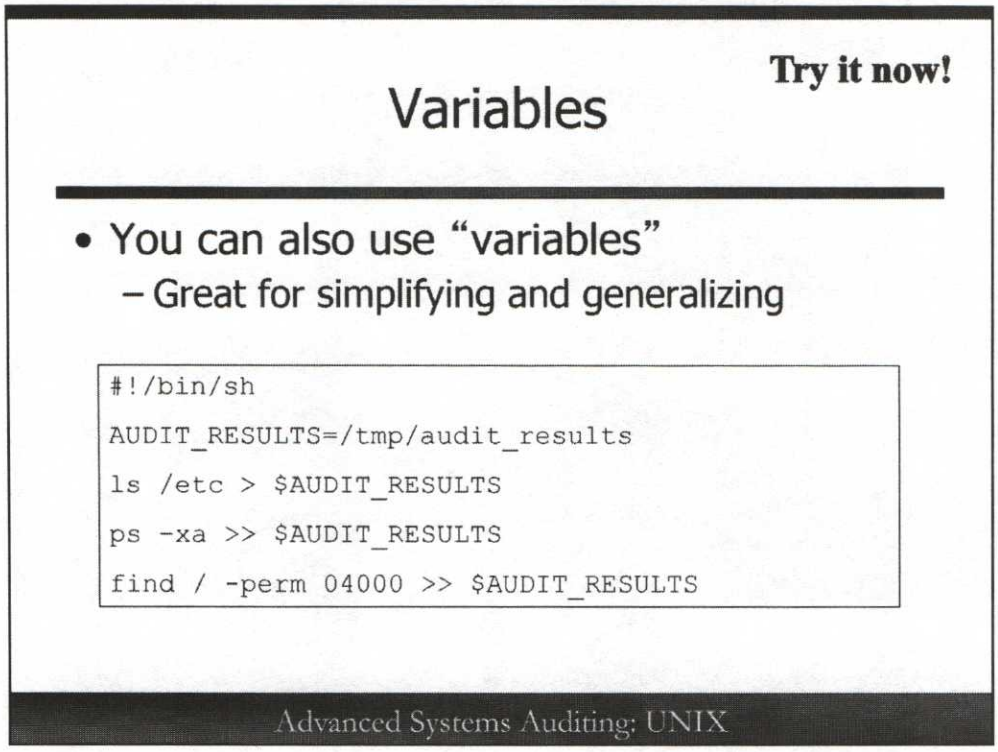
© 2016 David Hoelzer & Enclave Forensics™



UNIX scripting, particularly bash, csh, tcsh, sh, ksh, and other command-line interpreter scripting, is very typical of the command-line scripting languages for most platforms. If you have a series of commands that you want to run, simply enter them in line by line and save them into a file. In fact, this is why scripting is called "scripting"! We're simply writing a "script" that the computer should follow just like an actor in a play.

We can use all of the normal UNIX commands that are available, and we can even save our results by using output redirection (the > sign). It is very important to begin the script with the phrase "#!/bin/sh". This specifies the path to the shell that should be used to interpret the script. We could also use "#!/bin/bash" in this case, but bash and sh are typically equivalent these days.

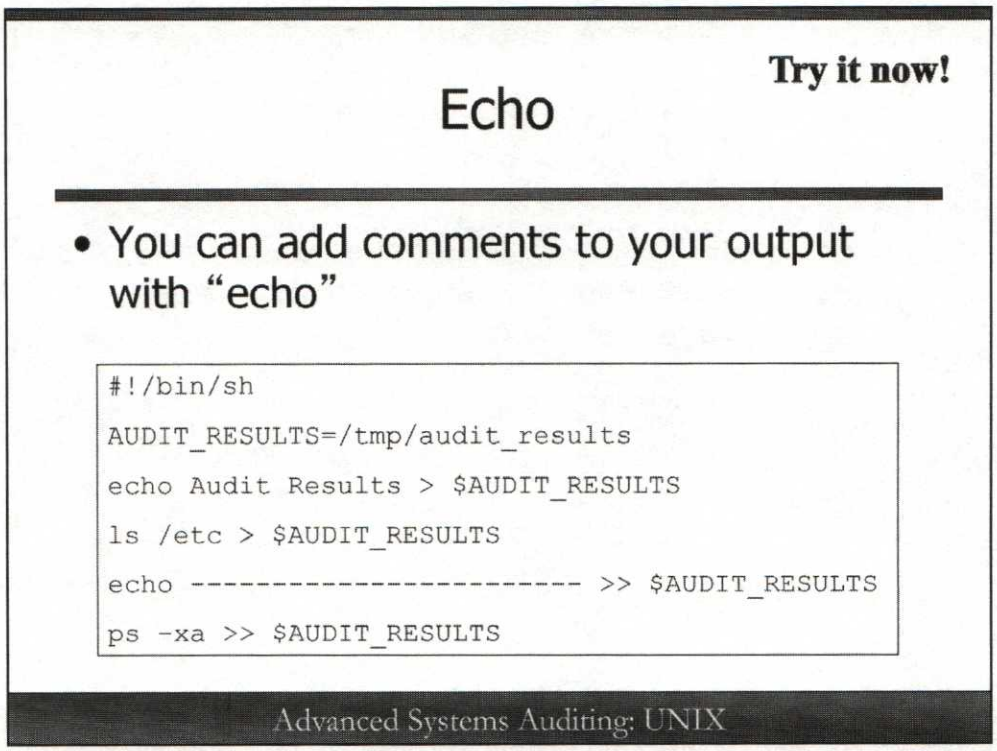
© 2016 David Hoelzer & Enclave Forensics™



One of the things that you will want to be able to do very early on in scripting is to generalize your script or make it easily customizable. If there is something that you do repeatedly in your script, for instance, storing the results into a specific file, then this is a perfect target for creating a variable. Notice that in the slide, we specify a variable called "AUDIT_RESULTS", storing in it "/tmp/audit_results". Now, every place that we used that filename in the previous slide, we can replace with $AUDIT_RESULTS.

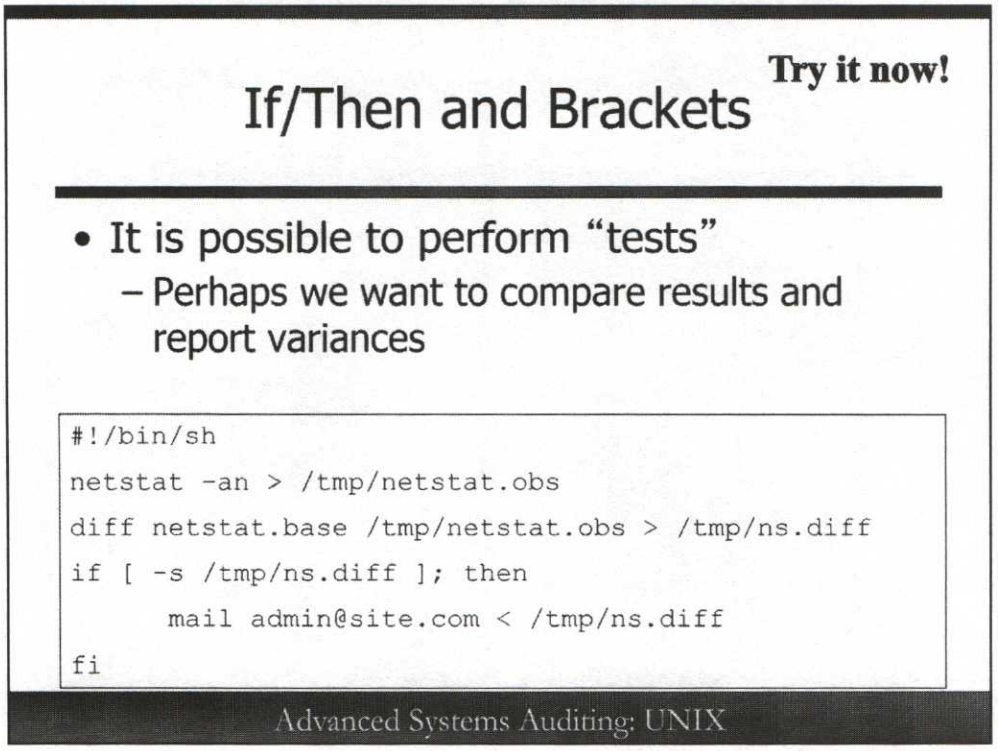
You might be wondering, "What's the point?" The real key here is that if you want to change the destination for the results, you no longer need to change every single line in the script. One change alters the whole thing! There are some other more advanced techniques for dynamically generating the contents of a variable. Depending on the skill of the class, the instructor might choose to cover some additional ways of handling variables.

© 2016 David Hoelzer & Enclave Forensics™



So far, our script simply dumps the output of every command into a file. Especially if we're gathering results from many machines, we need a way to distinguish the output from the various systems. To accomplish this, we can use the "echo" command. This command can be used to echo arbitrary strings into our file, too. We can use this to add the name of the host or perhaps the date or any other pertinent piece of information.

© 2016 David Hoelzer & Enclave Forensics™



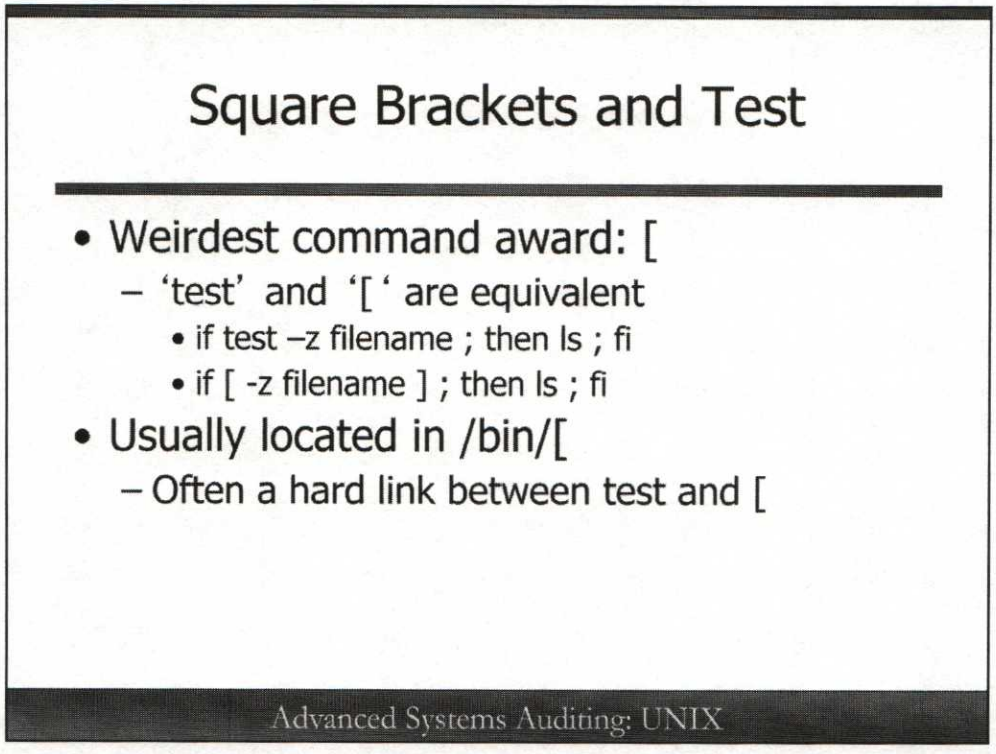
Especially if you're planning to use the script to perform automated security auditing and reporting, one of our goals is to get the administrators or security officers to look at the reports. If the system continually sends blank or meaningless reports, we will likely not reach our goal, so we might choose to test to see whether or not there were any meaningful results.

In the script above, we have previously collected baseline information for listening ports. Now, when the script is running, we generate an observed file. After the observed file is created, the script runs the 'diff' command to identify any differences between the two files. To use this to produce our report, we can set up a conditional.

The if/then structure is pictured above and is fairly self-explanatory, but the brackets need some explanation. The square brackets are used to perform tests. In this case, the '-s' is being used to test whether the file size of "ns.diff" is greater than zero. If the file is empty, the script continues. If the file contains anything, the contents are e-mailed to the administrator at the site.

Notice the spacing! Square brackets should be spaced away from everything. If they aren't, the shell will try to interpret them as part of an expression and you will have errors on your hands.

© 2016 David Hoelzer & Enclave Forensics™

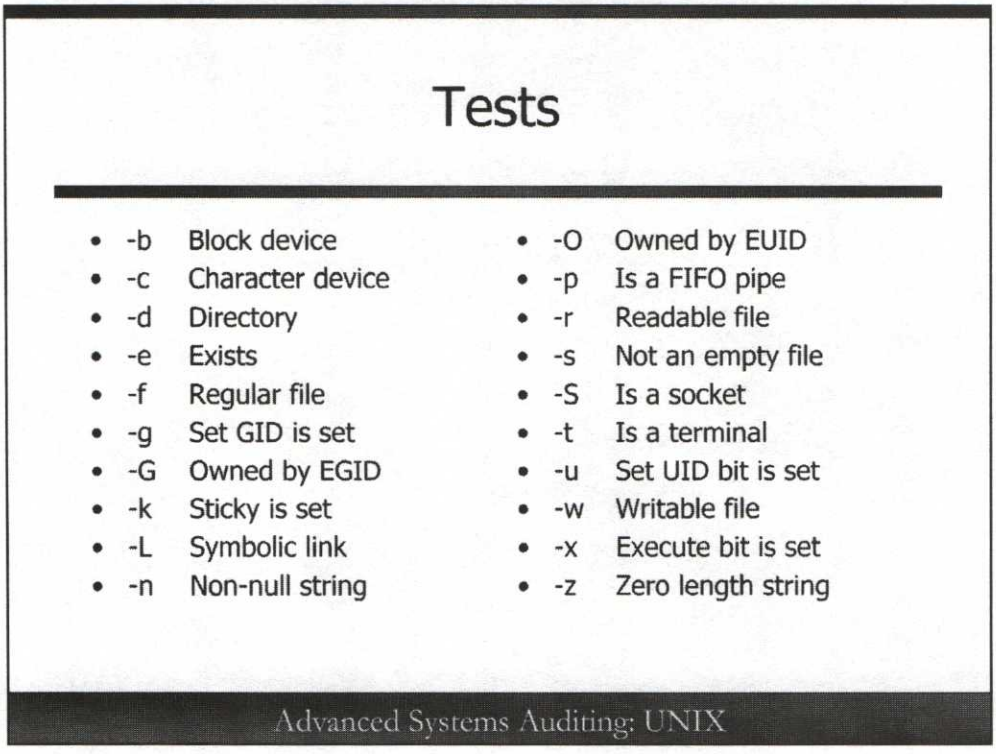


We should explain why the spacing around the brackets is so important. The brackets can be used for set definition ([a-z], for instance) when specifying filenames. If we press the [] up against the 'if', then the shell will assume that we are looking for a file named 'if[....]' rather than trying to create a conditional expression. The square bracket is actually equivalent to the 'test' command. They're so equivalent that on most systems, you'll find that they are actually the same file. For instance, notice in this example, the inode number:

```
david-hoelzers-imac:~ dhoelzer$ ls -li /bin/test /bin/\[
1374818 -r-xr-xr-x  2 root  wheel  46720 Nov 28  2007 /bin/[
1374818 -r-xr-xr-x  2 root  wheel  46720 Nov 28  2007 /bin/test
david-hoelzers-imac:~ dhoelzer$
```

The first number listed on the line in the directory listing is the inode number for the files. The inode is, in a sense, the file pointer. The filename is just an abstraction to refer to the actual inode, which points to the actual file content. As you can see, the 'test' and the '[' filenames are "hard linked," which means that they are two names for the same file. You could think of them as nicknames for the actual file. As an example, you are who you are no matter what name we use to refer to you. The same is true of the contents of inode 1374818 on the system used in the example.

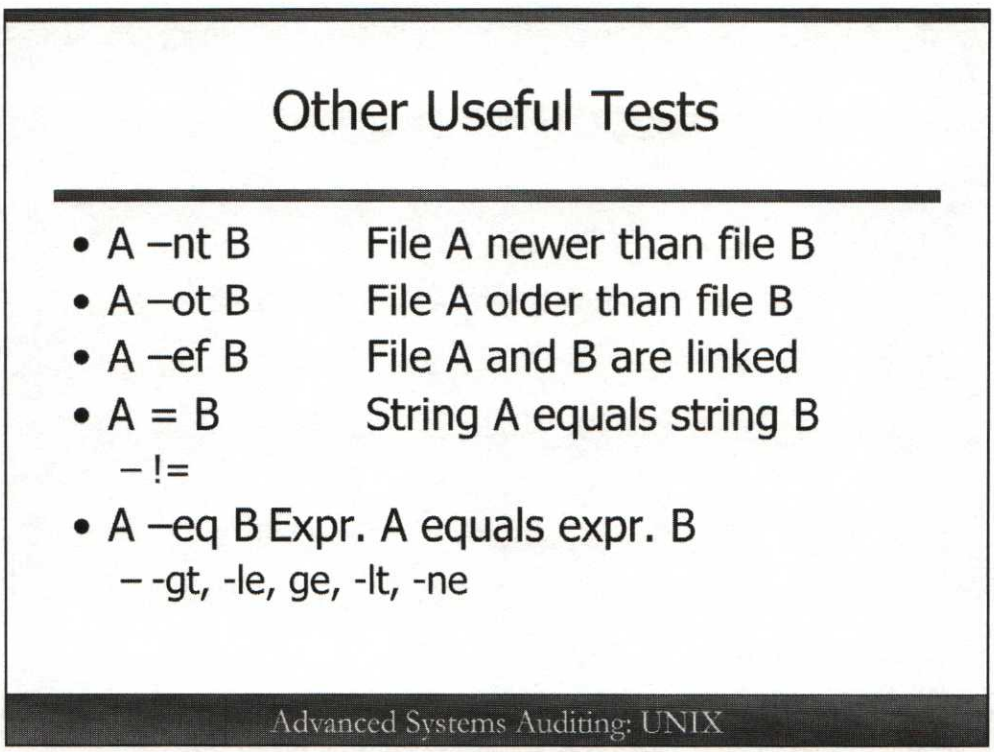
© 2016 David Hoelzer & Enclave Forensics™



This testing tool can be used to test for just about any condition that you can think of with regard to a file or directory. We can use it to test the type of file (directory, socket, block device, character, and terminal), examine the file permissions relative to the user executing the script (read, write, execute, owner, and group), test to see whether the file is empty, see whether a file exists, etc.

We're not limited to testing files, though that is the primary focus. We can also check the content of strings, looking to see whether they are empty or have contents. Comparisons can be performed as well. We have some of the comparisons listed on the next slide for you.

© 2016 David Hoelzer & Enclave Forensics™



The ability to see whether files are empty, exist, and so on, are very useful, but sometimes we want to perform a comparison. For instance, using the conditionals listed in the slide, we can test to see whether one file is older or newer than another, see whether two files refer to the same file (like test and [), and so on.

Again, although the primary focus is on files, we also have the ability to use conditionals on strings. These are very straightforward. Strings are either equal or not equal. On the other hand, we might want to perform some type of numerical expression and act on the results. Let's say we wanted to see which of two directories had more files in it. We could do something like this:

```
ONE=`ls / | wc –l`
TWO=`ls /etc | wc –l`
if [ $ONE –gt $TWO ] ; then
        echo Directory one contains more files.
        exit 1
fi
echo Directory two contains more files.
exit 2
```

© 2016 David Hoelzer & Enclave Forensics™

Try it now!

Command-Line Arguments

- Allows for generalization
  - Perhaps specify the output file for your results

```
#!/bin/sh
if [ -z $1 ]; then
      echo You must specify an output file!
      exit 1
fi
echo Sending results to: $1
```

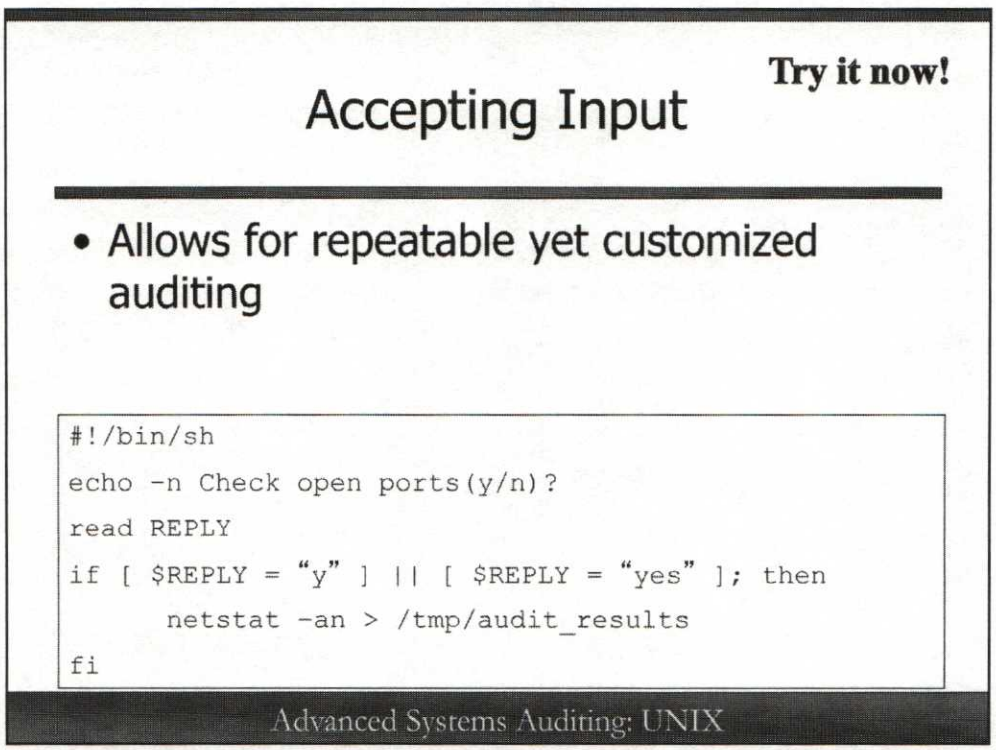
Advanced Systems Auditing: UNIX

Previously, we looked at using variables to customize or generalize our script. Command-line options give us another possibility for generalization. When you run your script, you can send command-line arguments. For instance:

./audit_script /tmp/results

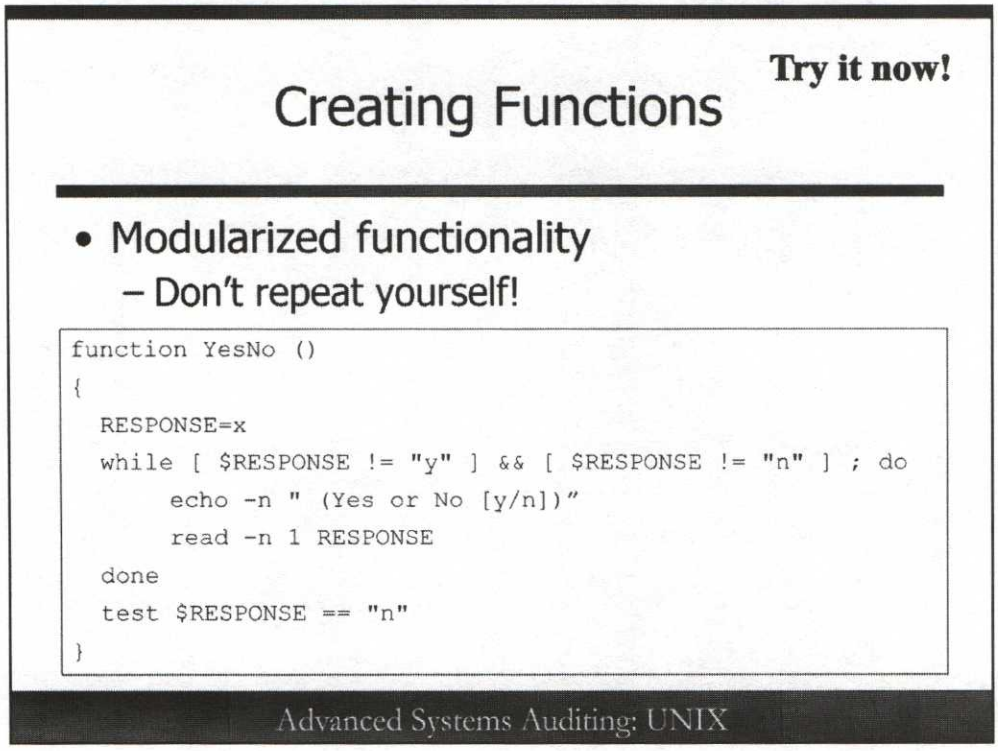
This attempts to run the script "audit_script" in the current directory sending the command-line argument of "/tmp/results". In the script, you can refer to this value using the variable "$1". Notice that we use a different test here, testing to see whether the variable "$1" is empty. For a list of other tests, please refer to the previous two pages for the kinds of tests that can be performed.

© 2016 David Hoelzer & Enclave Forensics™



You might also want to take input from the user while the script is running. Perhaps you have written a fairly comprehensive audit script that covers all aspects of a UNIX system. It might also be that your scope is somewhat limited when it comes to the system that you are running the script on. If this is the case, your script can be customized to allow you to choose which options to run on this particular system.

© 2016 David Hoelzer & Enclave Forensics™



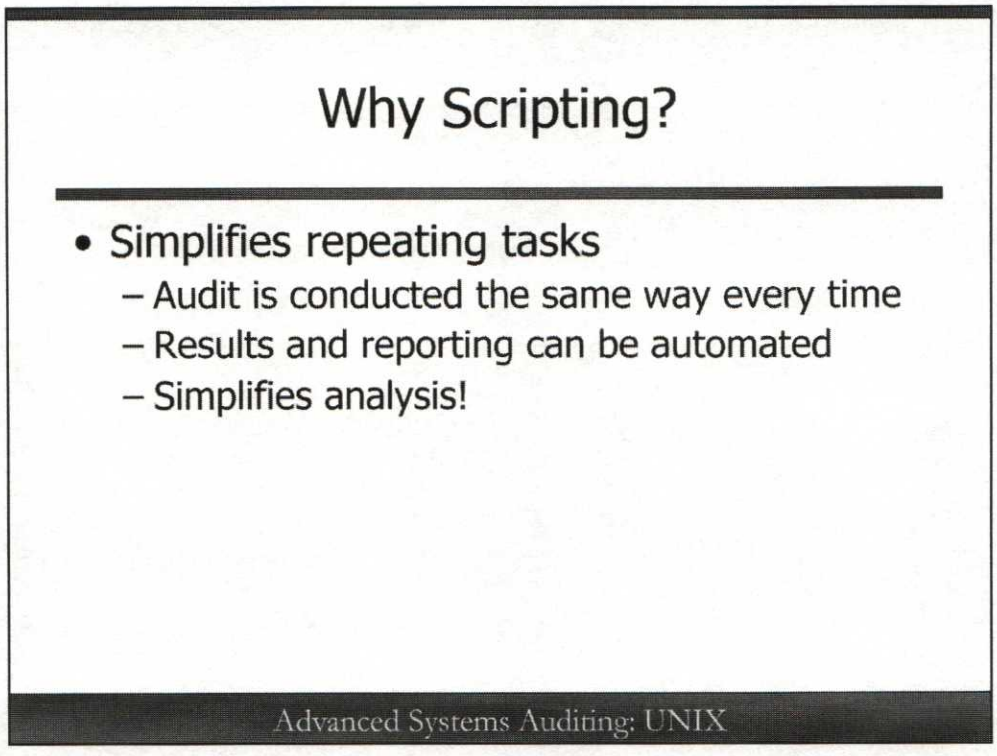
Probably one of the most important constructs in programming beyond conditionals are functions. A function is simply a collection of actions that you want to abstract out into a single step that can be repeated at any time.

Much like variables that allow us to define a value in one place and reuse it in many, thereby allowing us to centralize changes to values, functions allow us to define actions in one place and reuse them in many places. Now the location of the *functionality* is centralized, again allowing for easy and quick modification.

The coding mantra that can be really good to follow is "Don't repeat yourself!" If you find yourself performing the same activity a second time and that action is more than just a line or two of code, you should seriously think about how that functionality can be modularized.

The example in the slide abstracts out a "Yes/No" function. The second to last line in the script does deserve special mention, however. Here, we have a bare "test" statement and you might notice that there's apparently nothing relaying the results back to the code that called this function. The test function, in this case, will set the return code (which can be accessed using the built-in variable $?). If the response is 'n', $? is set to be 0. The only other possibility is 'y', which will result in $? being set to 1, allowing us to determine what happened in the function!

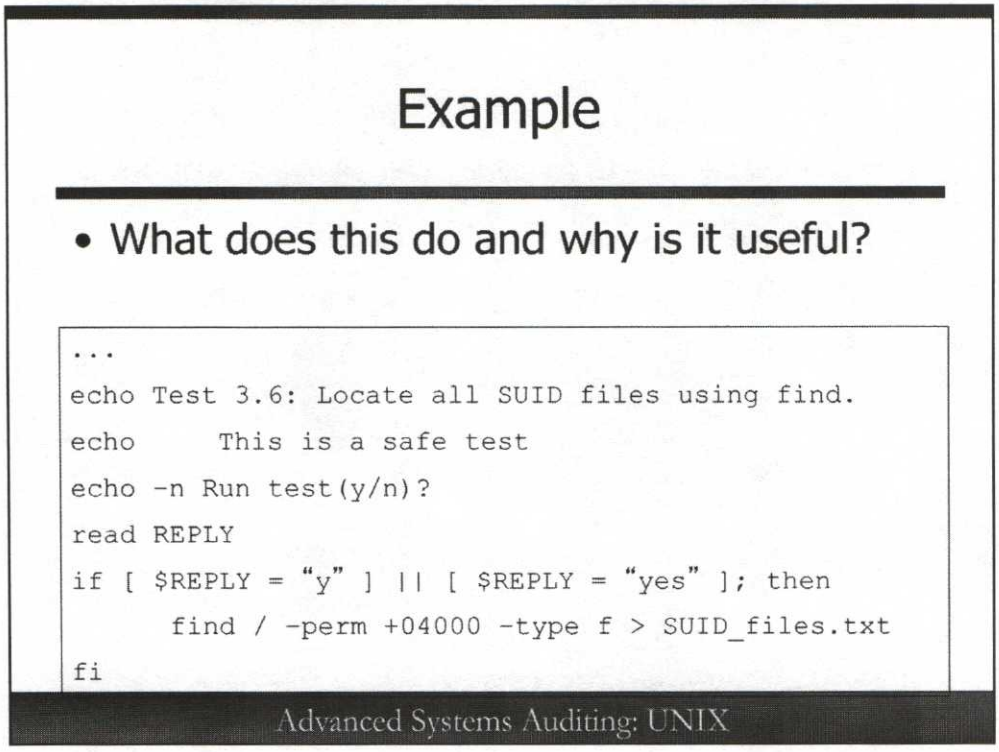
© 2016 David Hoelzer & Enclave Forensics™



We've covered only a very few basics, but even so we have enough ammunition to write some fairly complete audit scripts. Remember that our purpose in covering this material is not to make you a system administrator or to turn you into a programmer. Our real goal is to allow us to create repeatable audits and automated audits.

Scripts can also be very useful when it comes to performing analyses. For instance, earlier in the week, we looked at performing a firewall validation that would produce lots of output into a text file. Rather than trying to analyze all of that output by hand, we can write a very simple script to look for specific items or perhaps to summarize the results.

© 2016 David Hoelzer & Enclave Forensics™



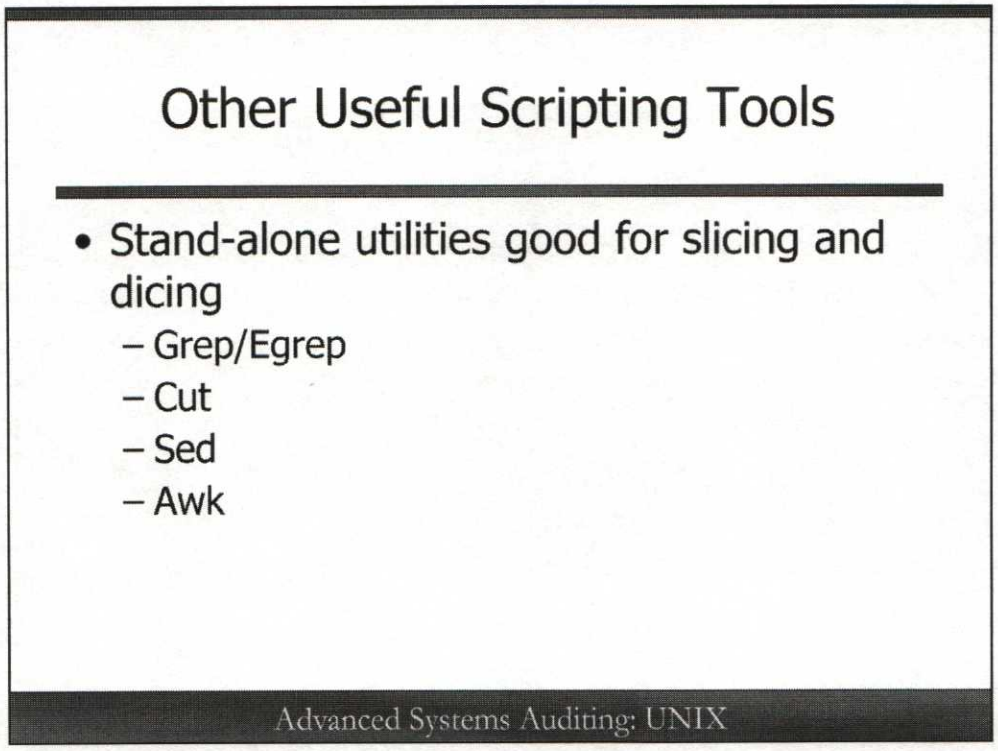
Look at the example in the slide for a moment. This is just one piece of a much larger script. Looking at what's listed, can you explain what this script does? At this point, even if you've never done any UNIX scripting in your life, you should be able to explain almost everything shown.

A more difficult question but an extremely important one is why would we ever want to do this? After all, can't we just type "find / -perm +04000 –type f > SUID_files.txt"? To script it, couldn't we just put that line into the script and leave out all of the other things listed? Of course, the answer to both of those questions is "Yes." So then why would we write something like this?

The reason is what we said several days ago in this class. Whenever possible, we will prefer to have the administrator actually run the tests and do the typing. We'd really like to avoid touching the systems ourselves but would rather we just observe what's happening and record results. By writing the script in this way, we can codify our audit checklist. By including questions asking whether or not each command should be run, we create something that we can put into the hands of the administrator. Now when the tests are run, we know that they will be run exactly as expected. We know that there won't be any typographical errors, we know what order the information will appear in, we know there won't be any *other* commands run. Additionally, if the administrator objects to something, he has the opportunity to discuss it with us first and perhaps even decline to run that particular test.

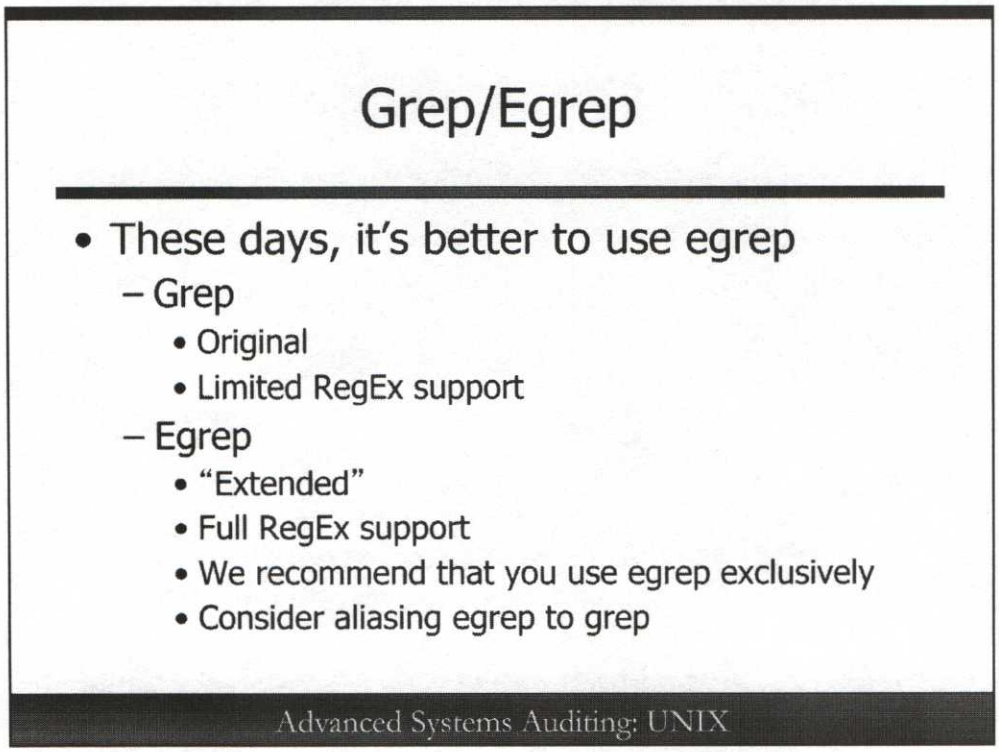
(This script prints out three lines of information with a question, "Run test(y/n)?" After this, it reads the user's response into the variable REPLY. Next, it checks to see whether the user has entered "y" or "yes." If the user has said "yes," it will finally run the "find" command with a specific set of options.)

© 2016 David Hoelzer & Enclave Forensics™



Although we're introducing the topic of scripting, let's also introduce you to a few other handy tools for slicing and dicing the output that comes out of the various other tools that we will use to collect information from our systems. All of these tools could easily take up an hour or more of our time, so we' re going to give you a very brief introduction to them over the next few slides and suggest some handy ways that they might be used. As we go through the material today, we'll use these tools to build up a useful information collection script.

© 2016 David Hoelzer & Enclave Forensics™

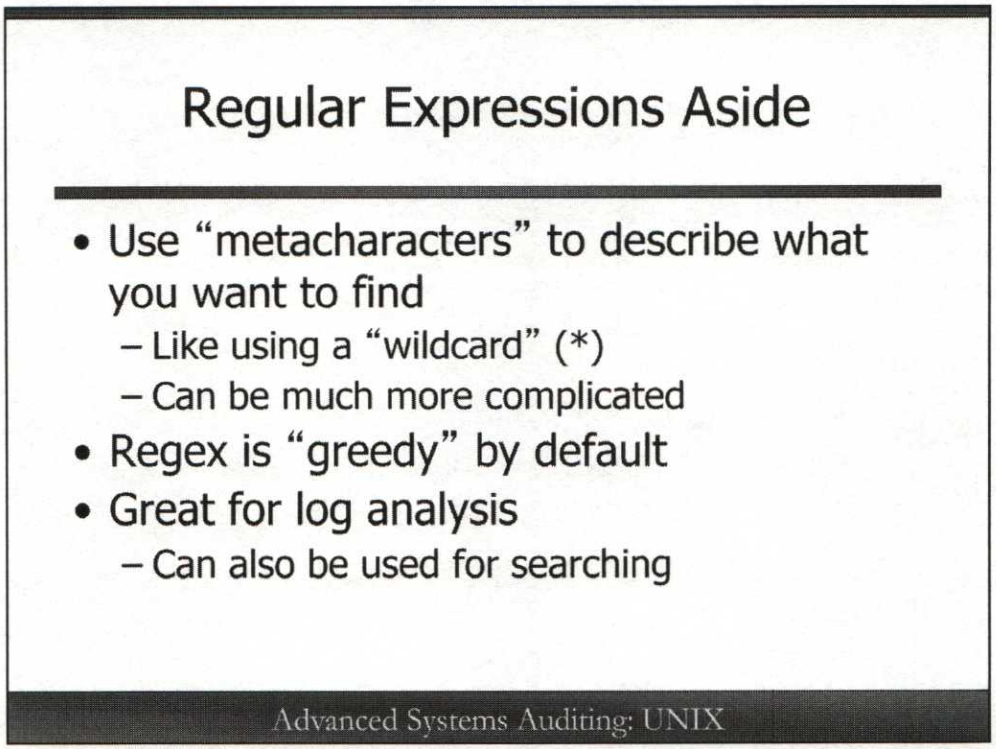


You should know that although everyone talks about using grep to find things, many administrators have switched to using egrep for any sort of substantial searching. Grep is the original tool, the name of which stands for "Get Regular ExPression."

The original grep has a somewhat limited regular expression library that it supports. Actually, it might be more accurate to say that grep supports the original expression set and egrep supports an extended set. Most notable in the egrep library is better support for character classes, character counts, back references, and more. We will look at some of these features as we work our way through the material today, working to put together a baseline audit script. Before we dig in any further, however, we should take a few minutes to discuss regular expressions themselves.

Generally speaking, it is recommended that you consider using egrep for everything and just forget about grep. In fact, I will usually alias the grep command to point at egrep. By doing so, I can overcome years and years of 'grep' habit and always make sure I'm using the expanded regular expression language supported by egrep.

© 2016 David Hoelzer & Enclave Forensics™

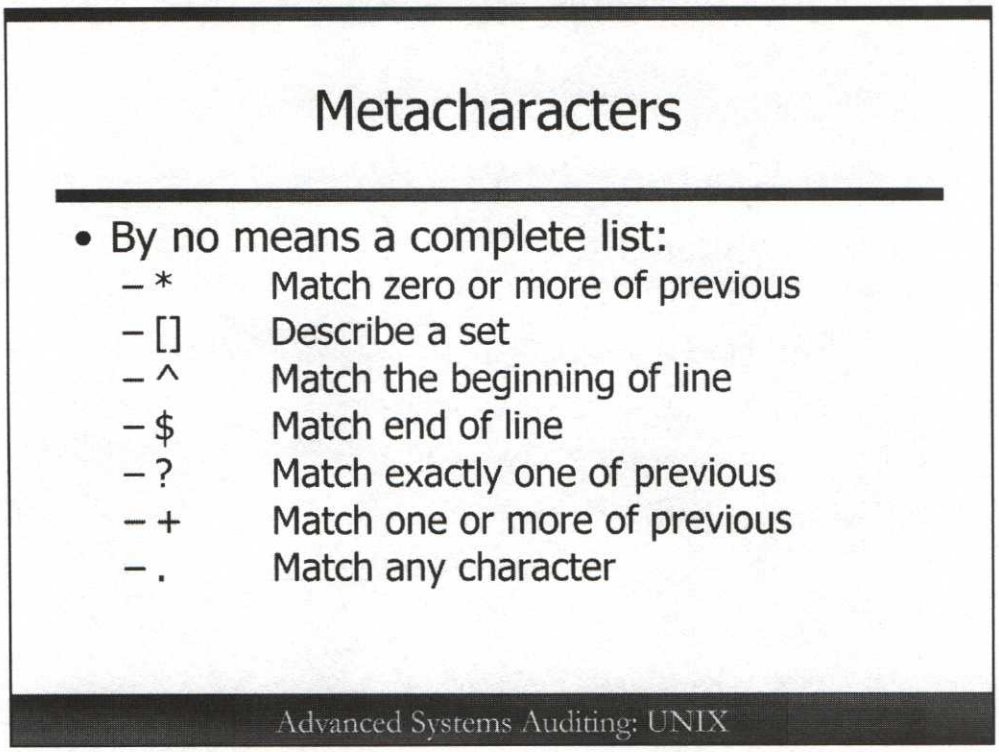


Another important capability within UNIX systems that can be used in connection with the 'find' utility and that can also be applied to log analysis and other purposes is regular expression matching.

Regular expressions make use of special tokens called "metacharacters" to describe patterns of characters that we would like to extract or identify in a text file or other data source. You can liken this to using the asterisk as a wildcard when getting a directory listing, but these are far more complicated and far more flexible.

An important fact to know about regular expressions is that, by default, they are greedy. What this means is that they will always match as soon as possible and as much as possible. This is important because it might seem that REGEX should find the best match, but this is simply not the case.

© 2016 David Hoelzer & Enclave Forensics™

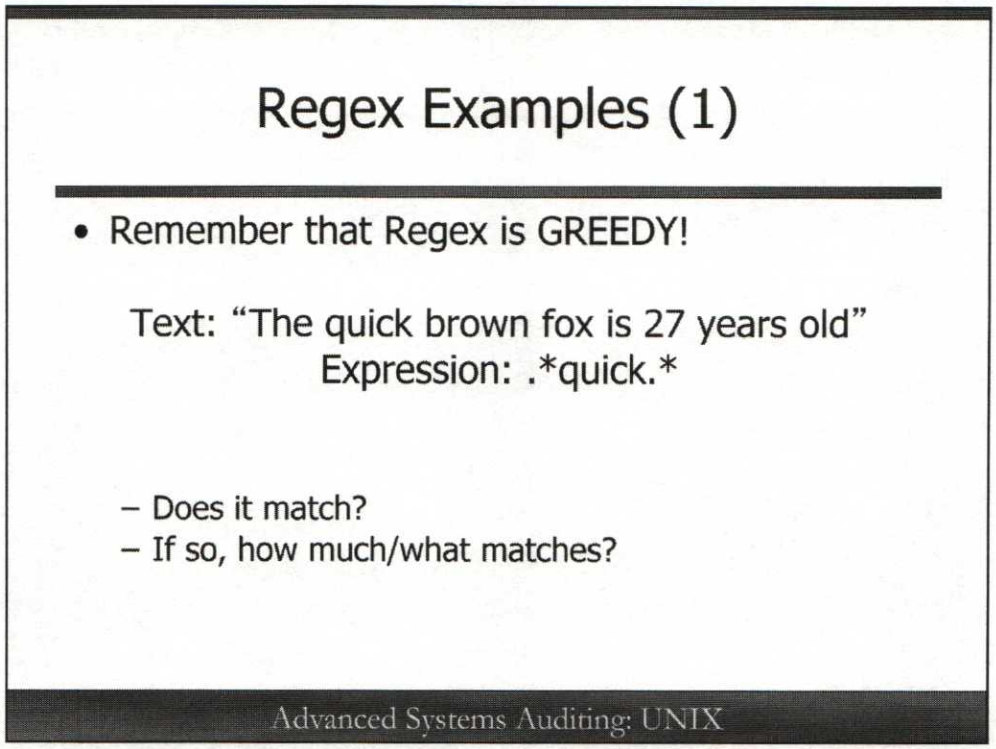


Let's start with these few metacharacters listed in the slide. This is by no means a complete list! There are many metacharacters. Even the characters list here can change meanings depending on the context that they are used in.

Of great importance is the fact that some of these characters are used to match zero, one or more of the *previous* character. This is very important because in other contexts, the asterisk is used to match anything, not just something that matches the last expression.

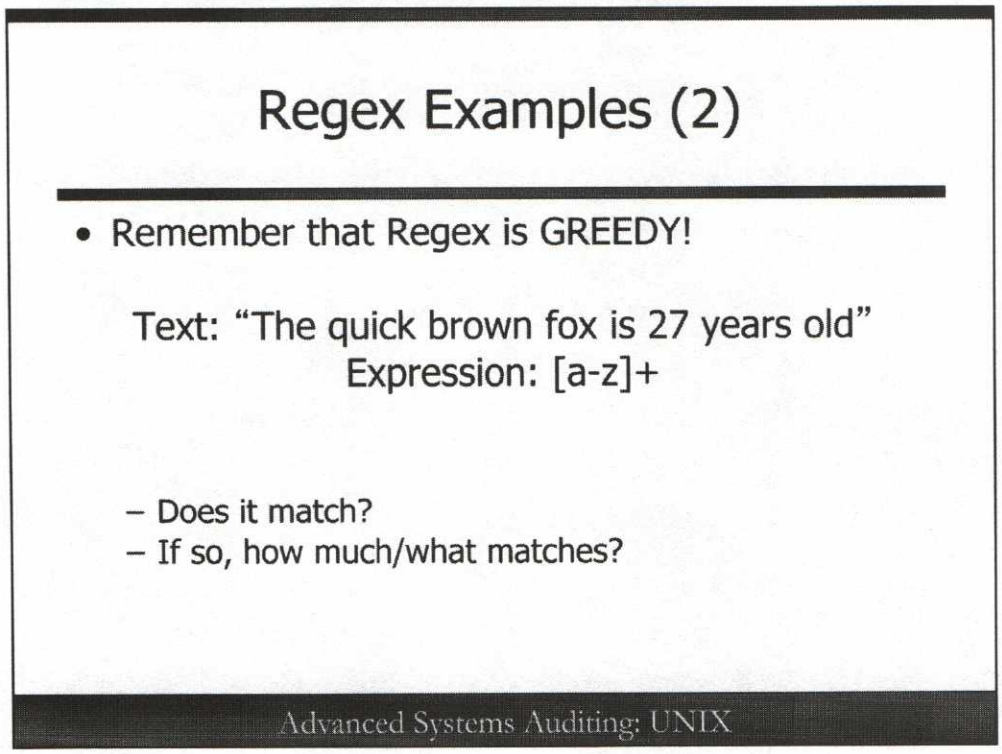
We can also combine these. For instance, to match zero or more of any character, the expression would actually be ".*". To match one or more, we would use ".+". Over the next few slides, the instructor will help the class to appreciate what the regular expression matches in the examples given. If you are working with this material at home, the very last slide in the book has the answers. No cheating!

© 2016 David Hoelzer & Enclave Forensics™



(The answers are in the back of the book right before the appendix—no cheating!)

© 2016 David Hoelzer & Enclave Forensics™



(The answers are in the back of the book right before the appendix—no cheating!)

© 2016 David Hoelzer & Enclave Forensics™

Regex Examples (3)

- Remember that Regex is GREEDY!

Text: "The quick brown fox is 27 years old"
Expression: ^[A-Za-z]*[0-9][a-zA-Z]*

- Does it match?
- If so, how much/what matches?

Advanced Systems Auditing: UNIX

(The answers are in the back of the book right before the appendix—no cheating!)

© 2016 David Hoelzer & Enclave Forensics™

Regex Examples (4)

- Remember that Regex is GREEDY!

Text: "The quick brown fox is 27 years old"
Expression: ^[A-Za-z]*[0-9][a-zA-Z]+

- Does it match?
- If so, how much/what matches?

Advanced Systems Auditing: UNIX

(The answers are in the back of the book right before the appendix—no cheating!)

© 2016 David Hoelzer & Enclave Forensics™

Regex Examples (5)

- Remember that Regex is GREEDY!

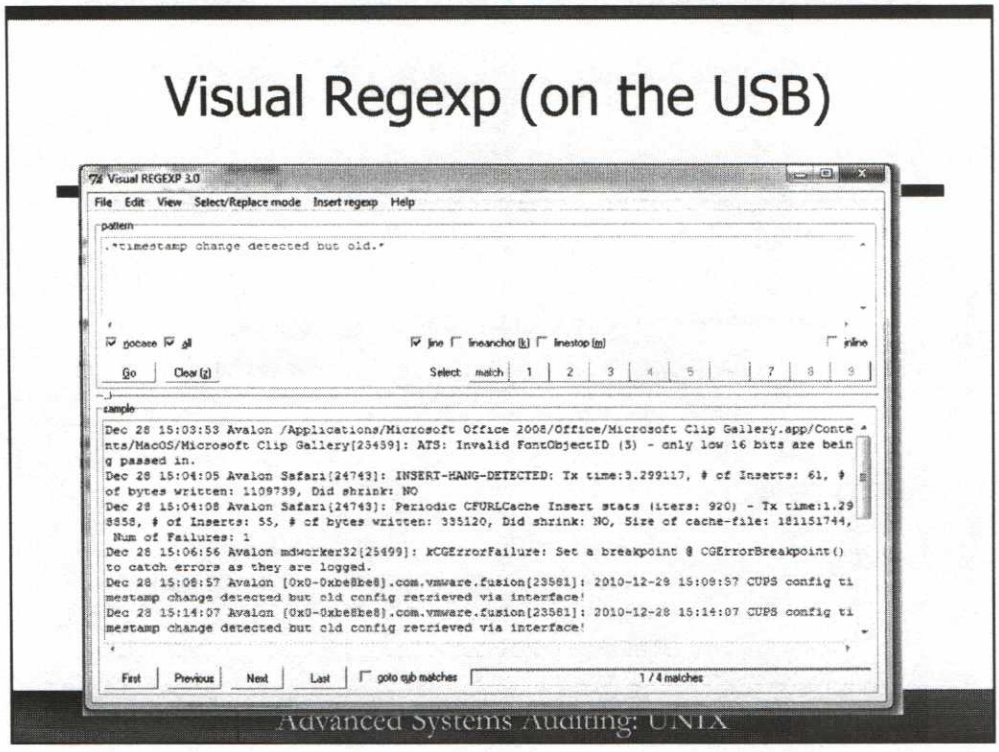
  Text: "The quick brown fox is 27 years old"
  Expression: ^[^0-9]+[0-9]+[^0-9]+$

  - Does it match?
  - If so, how much/what matches?
- Answers are on the last page of this book

Advanced Systems Auditing: UNIX

(The answers are in the back of the book right before the appendix—no cheating!)

© 2016 David Hoelzer & Enclave Forensics™

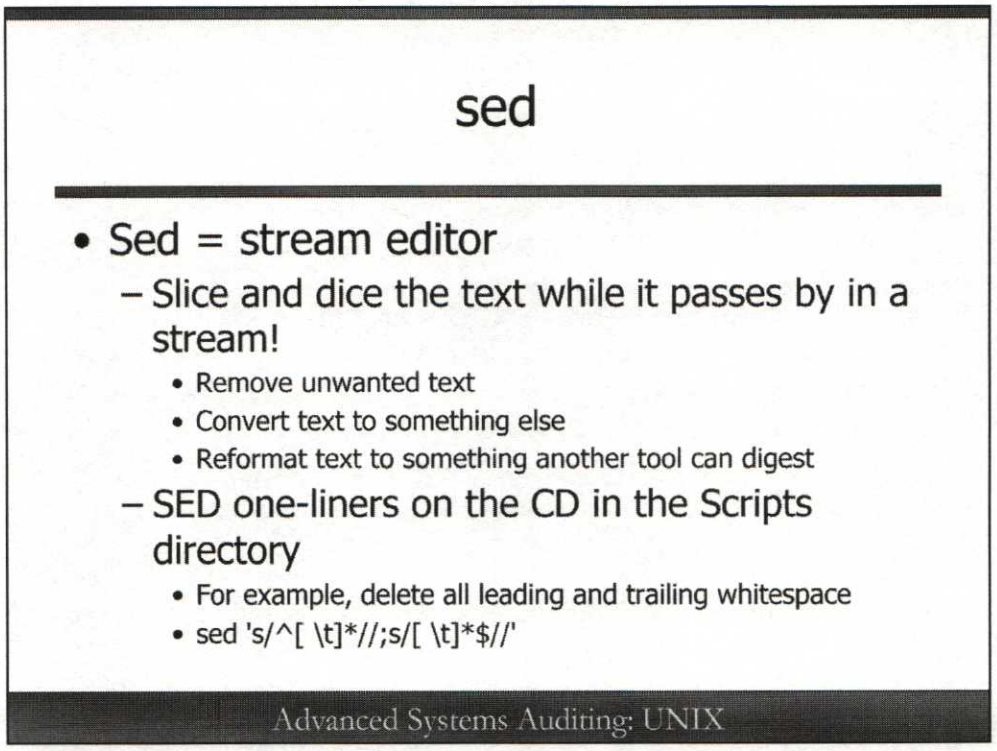


Whether you are new to regular expressions or an old hand, you might find the Visual Regexp tool extremely useful. You can find this Windows-based tool on the course USB in the "Tools" directory.

Aside from using this to test out regular expressions that you are attempting to create, this tool is exceptionally useful when trying to manage log processing rules. Log processing rules are almost always written using regular expressions. In the lower window, you can paste a sample from your logs, and then use the upper window to interactively create regular expressions to select specific types of events.

The resulting regular expression that you come up with can be used directly as-is in any regular expression based tool. This includes tools like awk, sed, grep, and so on.

© 2016 David Hoelzer & Enclave Forensics™

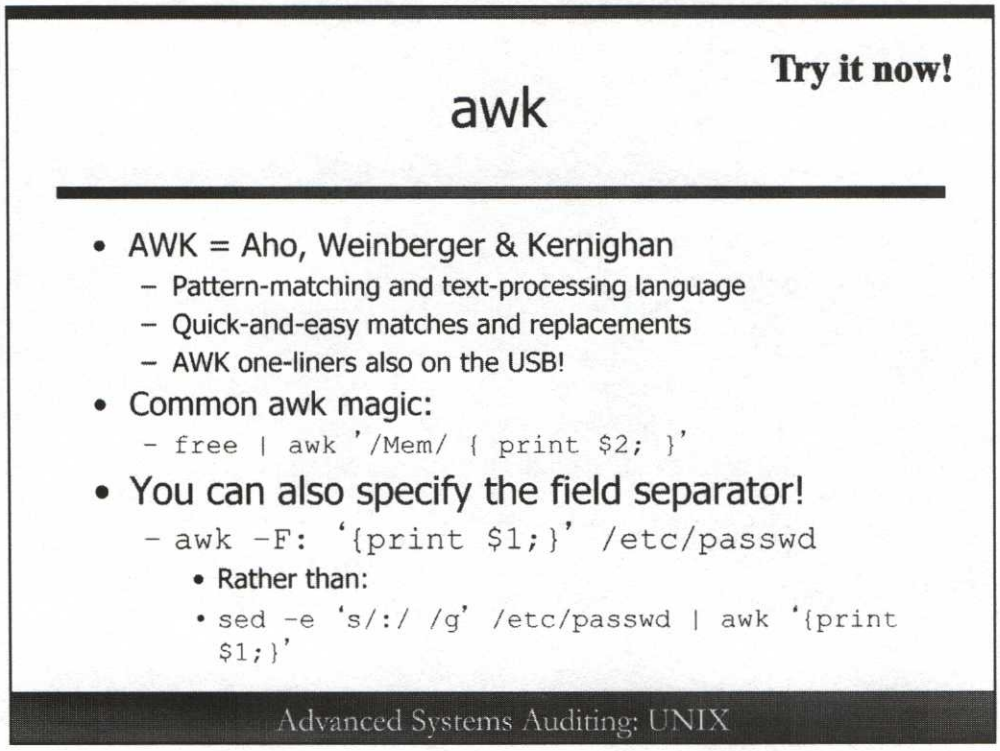


Our next tool is sed. Sed is the stream editor. It allows you to dynamically perform editing operations on streaming data one line at a time. Sed also makes use of regular expressions both for matching the text that interests you and for replacing or editing that text if that is your desire.

One of the most common uses for sed is to reformat text into the format expected by the next tool in the pipeline. For instance, to append a domain name to a list of host names, sed can do this for you pretty easily.

You will be interested to know that we have included the well-known list, "sed one-liners" on the CD in the scripts directory. It's worth your time to look at this list for useful quick tasks that you might need to accomplish before trying to create your own sed script.

© 2016 David Hoelzer & Enclave Forensics™



The last tool on our short list of tools to know is awk. Awk doesn't' t stand for anything more than the initials of the names of the creator of the pattern-matching language that it represents.

Although each tool has its specific uses, awk is more of a general purpose tool in that it can reproduce the behavior of several of the tools already mentioned. Actually, sed and awk are usually interchangeable, though it is best to be somewhat familiar with both. Although you can solve any text-formatting problem with either one, it is quite likely that a super complex sed expression will have a relatively simple awk counterpart and vice-versa.

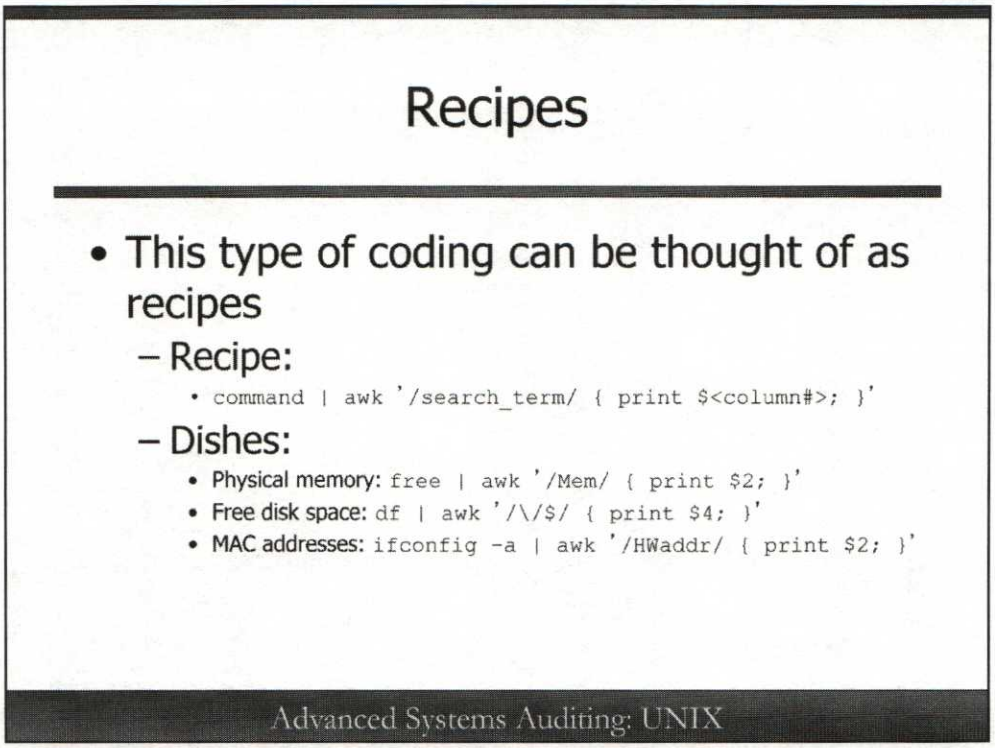
Another useful feature of awk is the ability to specify a field separator. Don't scoff! This can save you a lot of work. What this means is that rather than looking for whitespace, we can tell awk to use something else to split the words on a line. Without this ability, we'd be forced into some unpleasant sed-ness:

Sed –e 's/:/ /g' /etc/passwd | awk '{print $1;}'

This will accomplish (with a regular expression and sed) the same thing as the example in the slide.

As with the sed tool, we have included the well-known "AWK One Liners" text on the CD in the Scripts directory. Most common AWK tasks can be found in this file.

© 2016 David Hoelzer & Enclave Forensics™

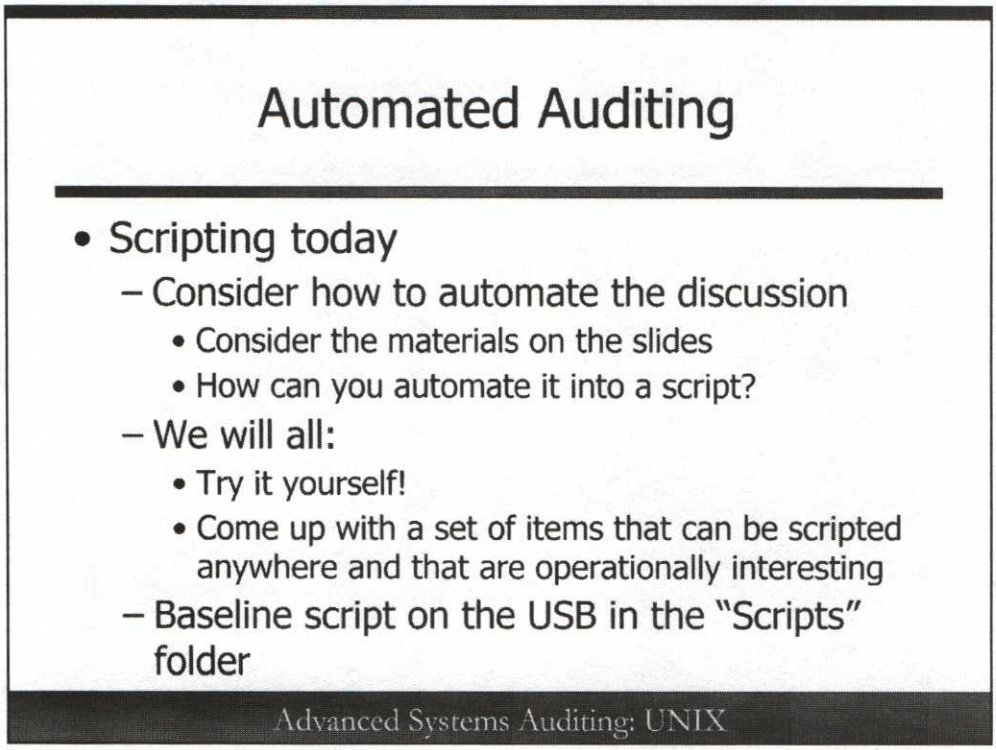


We'd strongly encourage you to take a step back and think about this in terms of "recipes" rather than focusing too hard on learning to write code. This type of scripting really doesn't require any advanced programming techniques or concepts. All that's really necessary are a few basics and a couple of interesting recipes.

For example, in the slide we list a recipe that can be used with the AWK tool to extract any column off any line in the output of any command. By generalizing out to a recipe like this, we can create a quick reference chart for ourselves and use that to create our script rather than having to learn all of the ins and outs of AWK.

The result is shown in the last three examples. Using that very simple recipe, we can extract out the amount of physical memory installed, the amount of disk space free, and the MAC addresses of all network adapters installed in the system. Of course, we can extract just about anything else that you can think of too!

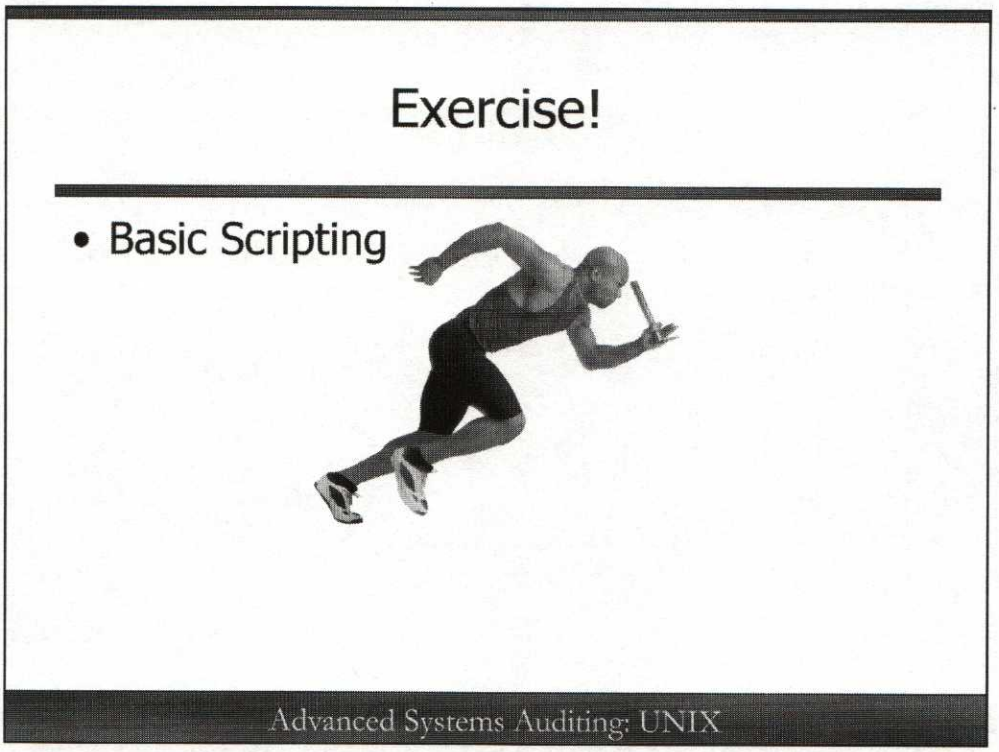
© 2016 David Hoelzer & Enclave Forensics™



In a few moments, you embark on a lab exercise to create a basic script. After that's done, we take a look at an example baseline audit script for UNIX machines that you can find on the USB and installed on the virtual machine that is used today.

From there on out, we will encourage you to consider which of the various tests would be useful for you to include in a baseline system and to also think about how to automate those things. As an auditor, we are interested in this from the point of view of simplifying our process and creating consistency. From an operational standpoint, we're interested in passing this knowledge on or working with an administrator to create this system so that we have sustainable systems providing important information and feedback to administrators!

© 2016 David Hoelzer & Enclave Forensics™



Please get started with the Scripting lab in the workbook. At the end of the exercise period, the instructor will walk you through the included baseline checklist.

© 2016 David Hoelzer & Enclave Forensics™

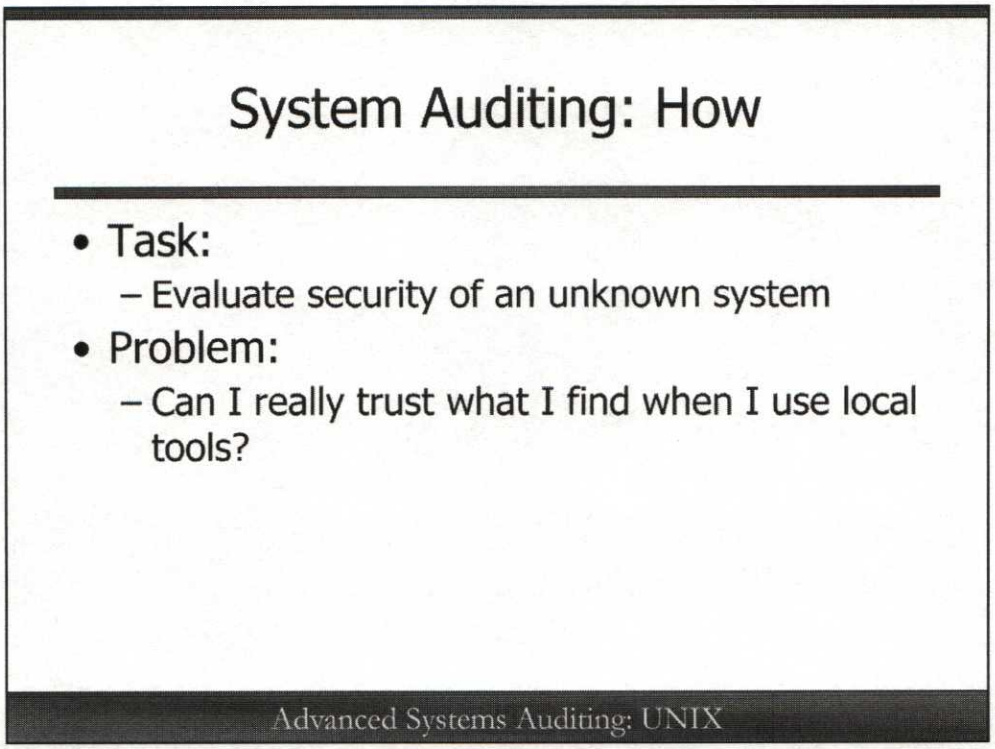
Roadmap

- Auditing UNIX
  - How
  - What
  - Why

Advanced Systems Auditing: UNIX

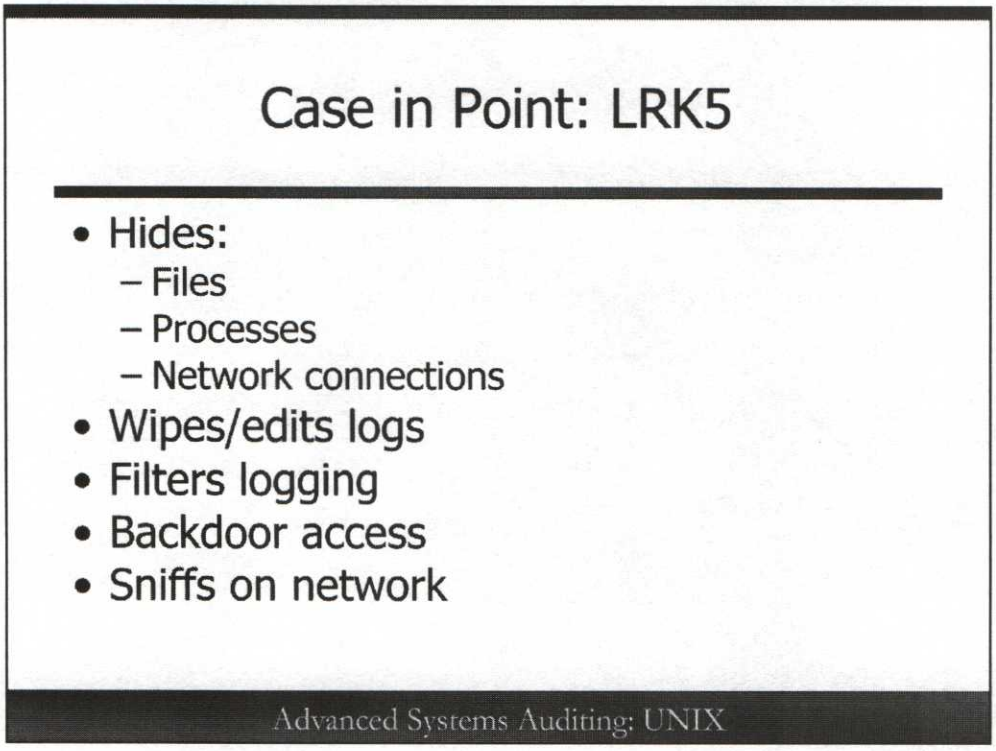
In this next section, we walk through a series of audit objectives and activities that will be used in the majority of UNIX audits. In the end, you will have a good framework of steps that can be taken to evaluate a UNIX system in your own environment.

© 2016 David Hoelzer & Enclave Forensics™



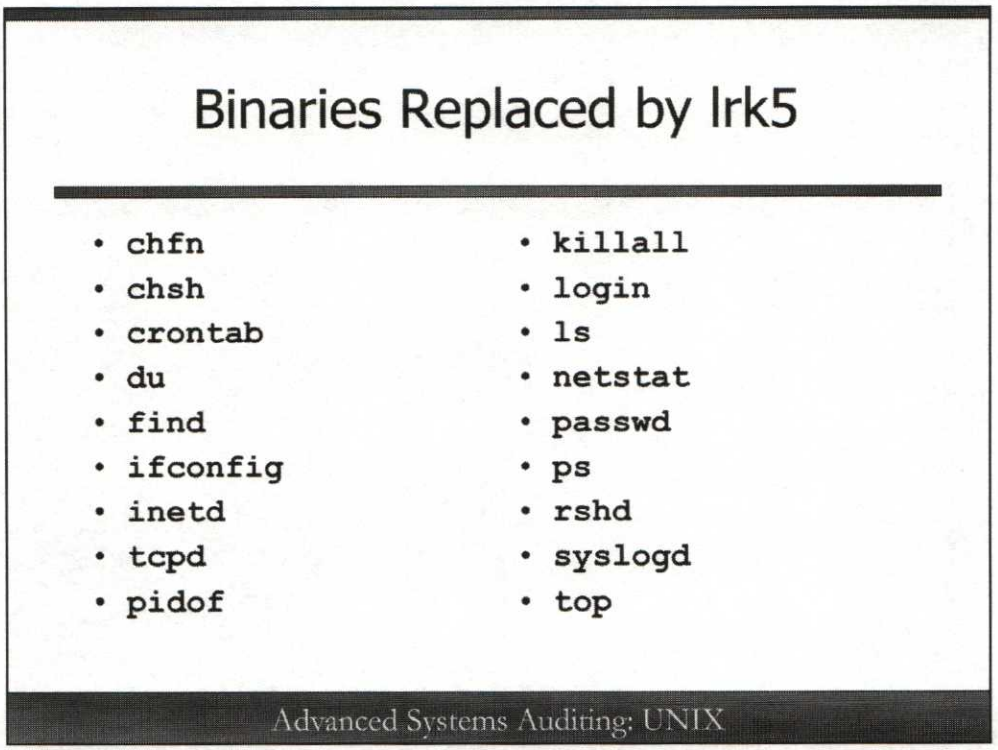
When approaching the audit of any system, there is a fundamental question that must be asked: Can I "trust" the information stored on the system and can I trust the tools that reside on that system? Attackers have become better and better at concealing their presence over time. One of the ways that they have done this in recent years is to create what's called a "Rootkit." This set of tools serves two purposes. First, the rootkit usually provides special access tools for the attacker, but secondly, the rootkit will serve to mask the fact that the system has been compromised. Let's take a closer look at one of these rootkits.

© 2016 David Hoelzer & Enclave Forensics™



Lrk (linux rootkit) version 5 incorporates these capabilities. Everything that the hacker needs to set up an invisible "base camp" where he can conduct covert operations…undetected and unmolested. This rootkit provides the ability to hide files and processes, as well as hiding selected network connections and services. The rootkit includes a log editor, allowing the attacker to mask his presence while leaving the logs in place, as well as filtering access entries that go to the logs after the rootkit is installed. To provide easy return access for the attacker, a concealed backdoor is installed. Finally, to allow the attacker to gather more information and perhaps to jump from this machine to another (lillypadding), a network sniffer is run.

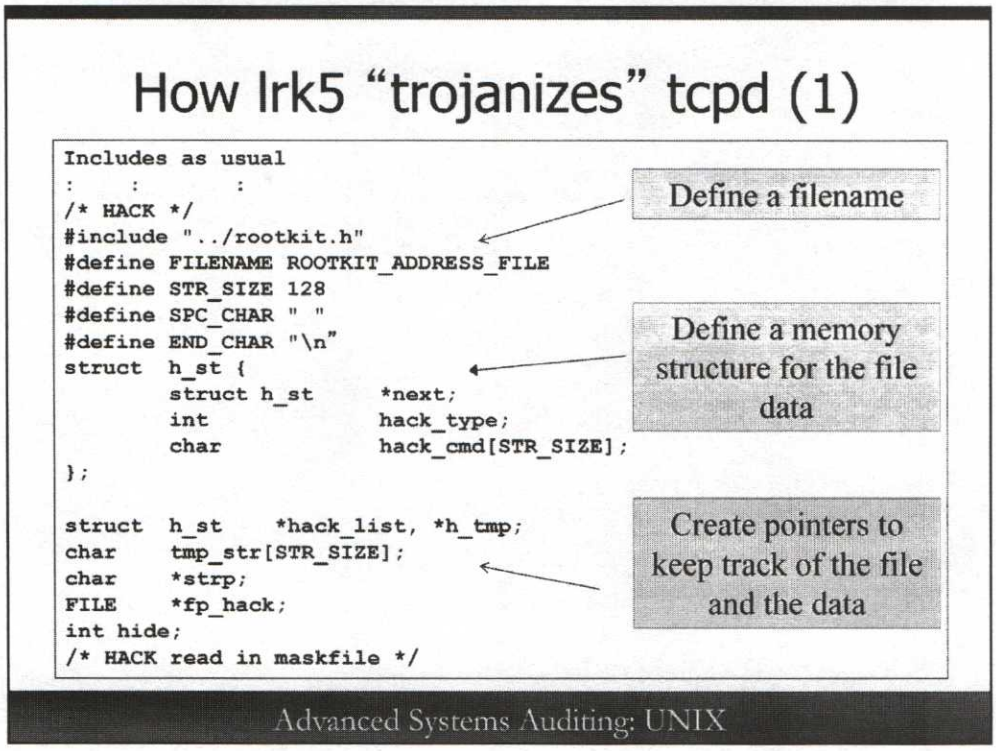
© 2016 David Hoelzer & Enclave Forensics™



The slide here lists just a few of the tools that are replaced by LRK5. Consider the problem of identifying whether or not a rootkit has been installed. You might start by checking which processes are running, but both 'ps' and 'top' have been replaced. You might check to see whether there are any unusual directory entries, but 'ls' has been compromised. You might check to see the running network services, but 'netstat' has been replaced. What about your logs? 'syslogd' has been replaced.

You can see the problem and the purpose of these tools. No matter how you might approach the problem of identifying the rootkit, the required tools have been replaced.

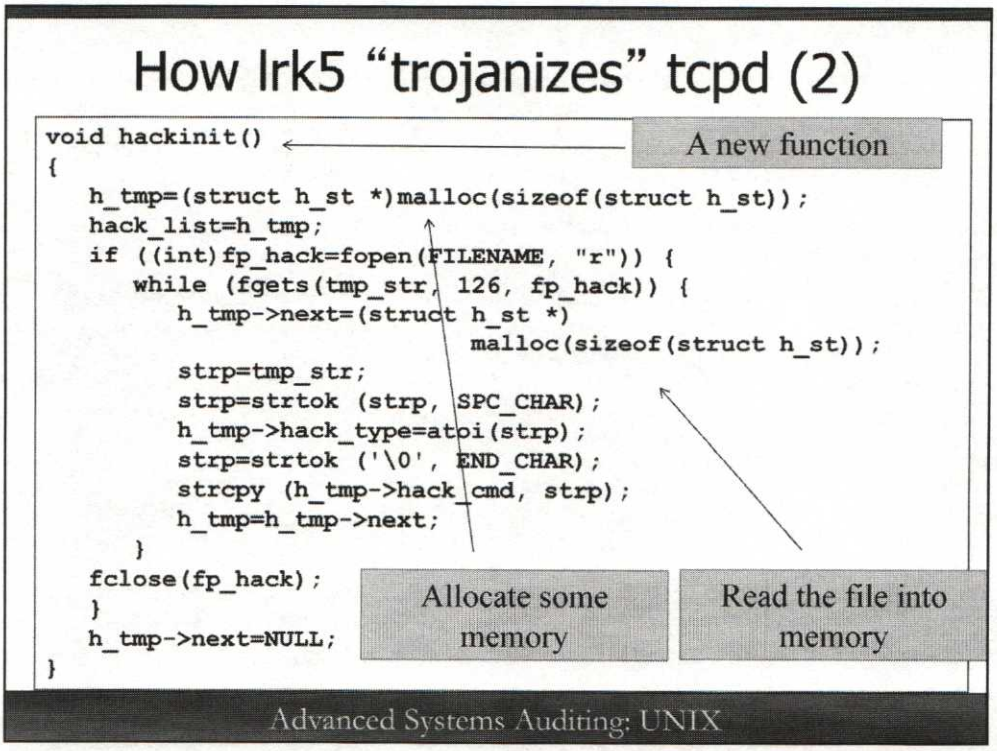
© 2016 David Hoelzer & Enclave Forensics™



Lrk5 even trojan-izes tcpd! Let's look at how it's done. Starting with the original code, additions are made that will eventually allow the hacker to access the host without being logged. Perhaps the most insidious part of this rootkit is that, although the attacker's connections will no longer be logged, all other connections will continue to be logged normally!

Here, we see some code that defines some data structures and variables. Notice that the developer of LRK has even included some handy comments, allowing us to more easily identify the hacked code. The most important things in this slide are the definition of a FILENAME, "ROOTKIT_ADDRESS_FILE", and the creation of a special data structure, h_st.

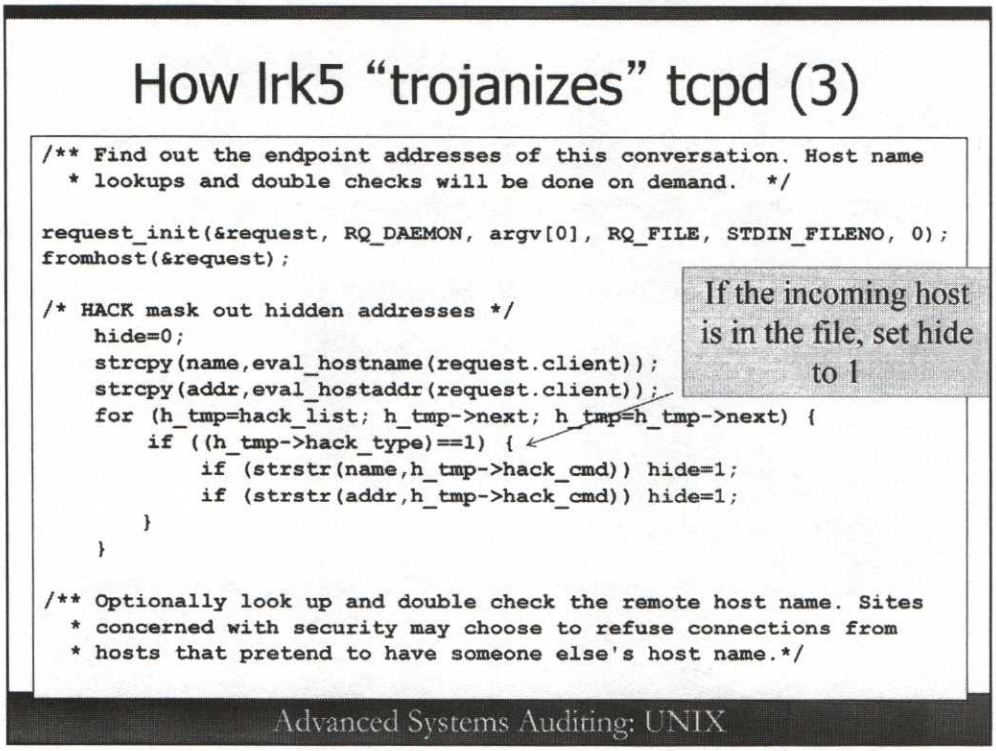
© 2016 David Hoelzer & Enclave Forensics™



Here, the code is reading and storing addresses that will eventually be allowed complete and unlogged access to the machine. This routine utilizes the h_st structure defined in the previous slide and populates it using the FILENAME, also defined on the previous slide.

Hackinit
allocate memory for a list of ipaddresses
if we can open a file containing ipaddresses then
while there are ipaddresses do
allocate memory for the next list element
get ipaddress from the file
move ipaddress into the list
setup link to next element
endwhile
endif
no dangling pointer!
endHackinit

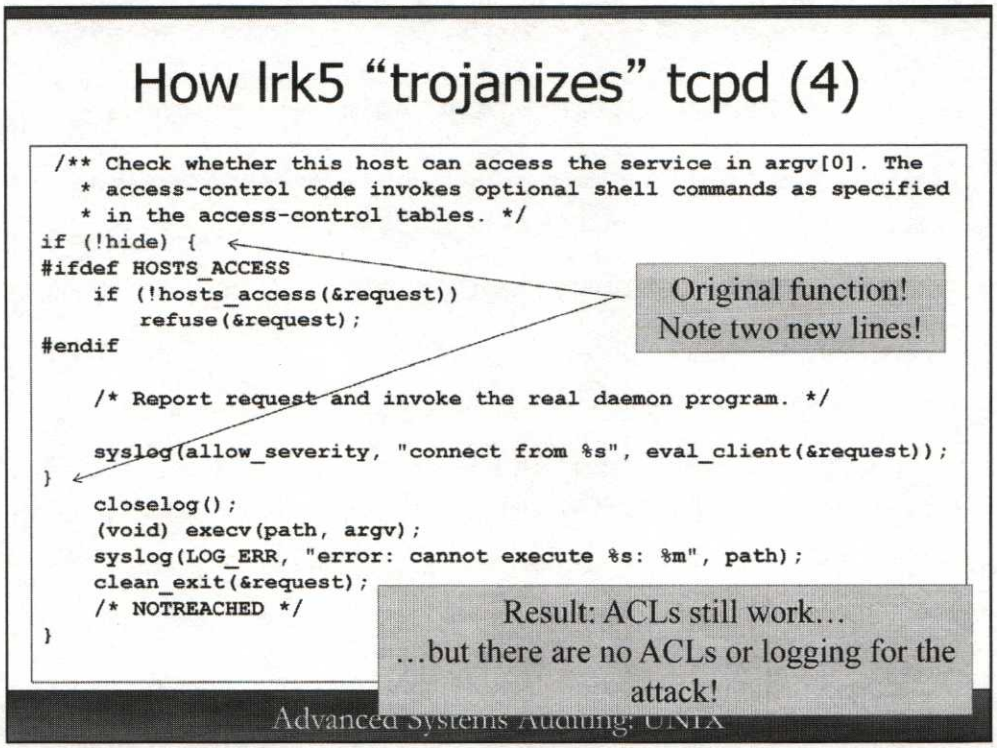
© 2016 David Hoelzer & Enclave Forensics™



Here's where the sneaky stuff really starts. The 'request_init' function is called every time a connection comes in from the network. Do you remember the 'hacklist' defined in the previous slide and populated from the predefined file? Whenever a connection comes in, it is compared against that list.

The code starts by setting the hide flag to false [hide=0]. Then, given the hostname or IP address of the connecting client, it runs through the "hacklist" of hosts that should be hidden. If the requesting client matches one of the IP addresses in the hacklist, then set the hide flag to true [hide=1].

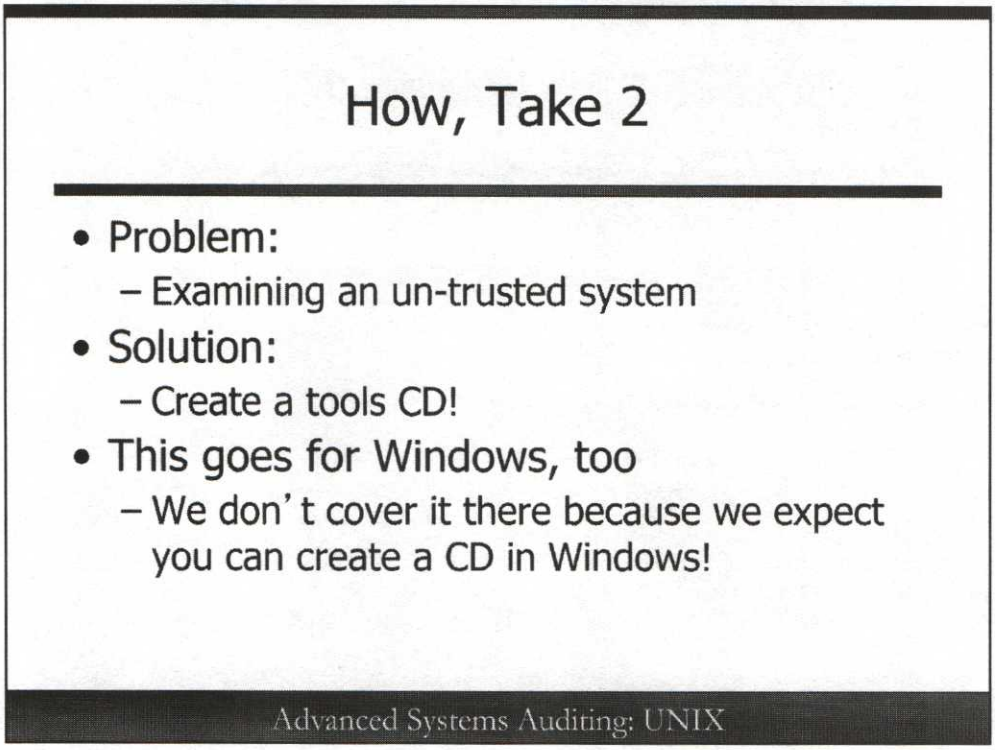
© 2016 David Hoelzer & Enclave Forensics™



And finally, this portion of code says that if it's not in the "hide" list, then process it in the normal fashion. Otherwise, we let the address bypass the Access Control List function and skip sending an entry to the syslog.

Note that most of the code of this hack is on the previous slides. However, the two lines of code on this slide are the ones that do the damage…that cause the ACLs to be bypassed. Everything up to this point has just been "good housekeeping."

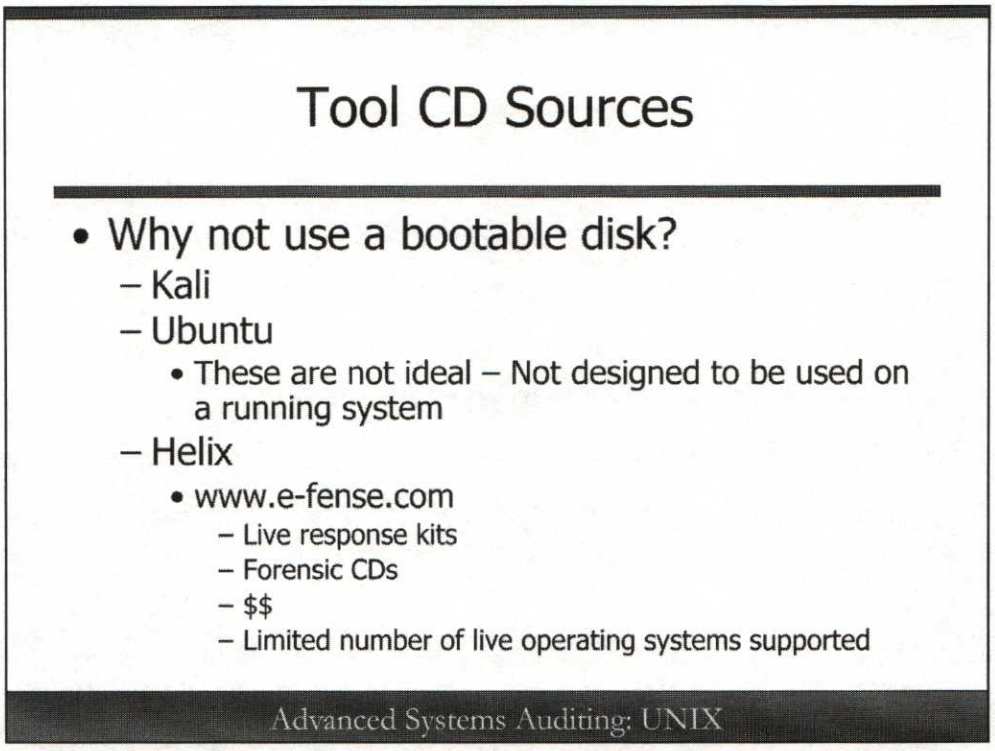
© 2016 David Hoelzer & Enclave Forensics™



So now that we can see why this is so important, let's get back to our original question: How can I evaluate a system if I can't "trust" the binaries on the system? The answer is, build yourself a CD with known good versions of all of the tools that you need!

This same concept (i.e., using a tools CD because we're not sure whether we can trust the system in question) applies for Windows. We don't spend any time with it on the Windows day because we expect that just about anyone taking this class can create a Windows CD and copy any needed executables onto the CD. In the UNIX world, though, this is not such a common task. In fact, many UNIX administrators aren't really sure how to create a CD of this type.

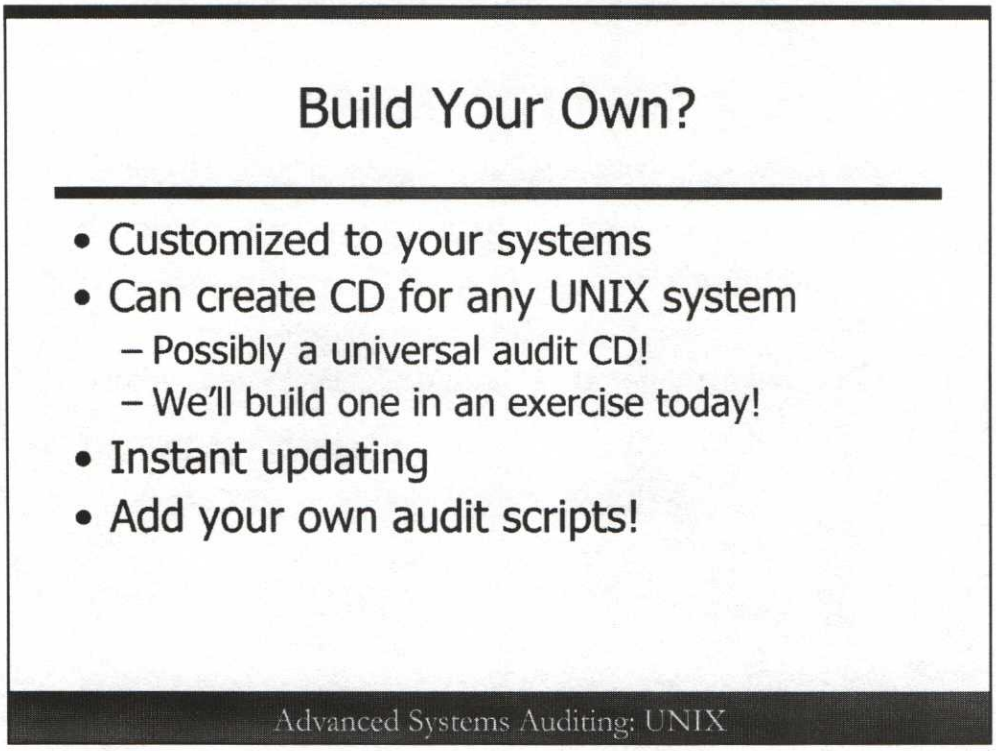
© 2016 David Hoelzer & Enclave Forensics™



There are some prebuilt sources that we can go to if we need to do work on Linux or OS X systems. For instance, we can obtain live CD systems for Ubuntu or backtrack that can be used with Linux and even OS X. The trouble here is that these systems are designed to be booted into, not to be mounted onto, a running system. This limits their usefulness for auditing purposes.

Another possibility is something like Helix. Helix is a very nice live response system from E-Fense.com. Currently, the system is useful but in transition as it moves from a free distribution to a subscription-based distribution. At the moment, the Helix distribution is focused more on acquisition of systems than analysis. This still works well for us in auditing because most of what we want to do is acquisition!

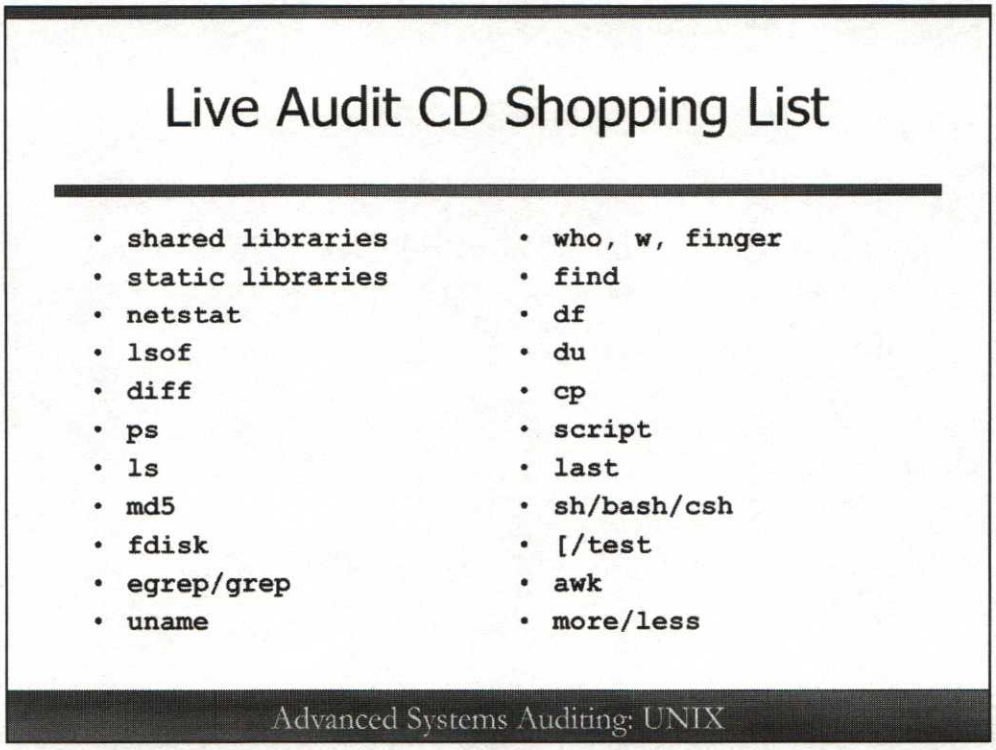
There is another problem though. What if you're using Solaris? What if you're using HP-UX or AIX? There is no live response system for these types of systems, so we have no choice except to build our own.



There are a number of advantages to building your own CD. First of all, you can customize the CD with the tools that you use, some of which might not be in a prebuilt distribution. If you have written any scripts to assist in automating your audit, you can include all of those here as well!

Another advantage is the ability to create a single CD that contains all of the tools for all of the different UNIX systems that you are responsible for auditing. If you have HPUX machines, you can create an HPUX directory. If you have Solaris machines, you also have a Solaris directory, and so on. This gives you a universal audit CD on which you can store your tools and your baselines.

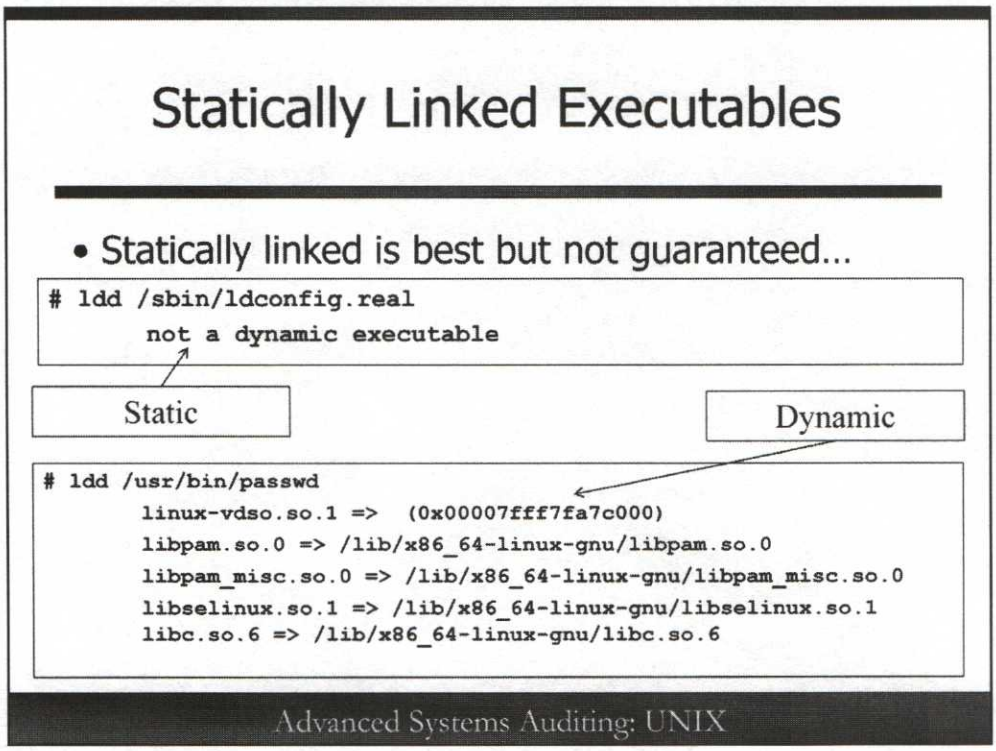
© 2016 David Hoelzer & Enclave Forensics™



For our live CD, we will want pretty much any and every tool that you might want to use to examine a system. This includes, of course, your scripts and whatever script interpreters are required. It also includes tools like "find" and "more," "ifconfig," and any other command that you can reasonably predict that you will need to execute during your engagement.

Generally, after you come up with a shopping list that works well for you, you can use that shopping list on any UNIX system that you run into. This means that although we will need to copy a different version of these tools over for every OS version that we will work with, it will be the same set of tools regardless of the operating system that we are working with on a particular engagement.

© 2016 David Hoelzer & Enclave Forensics™



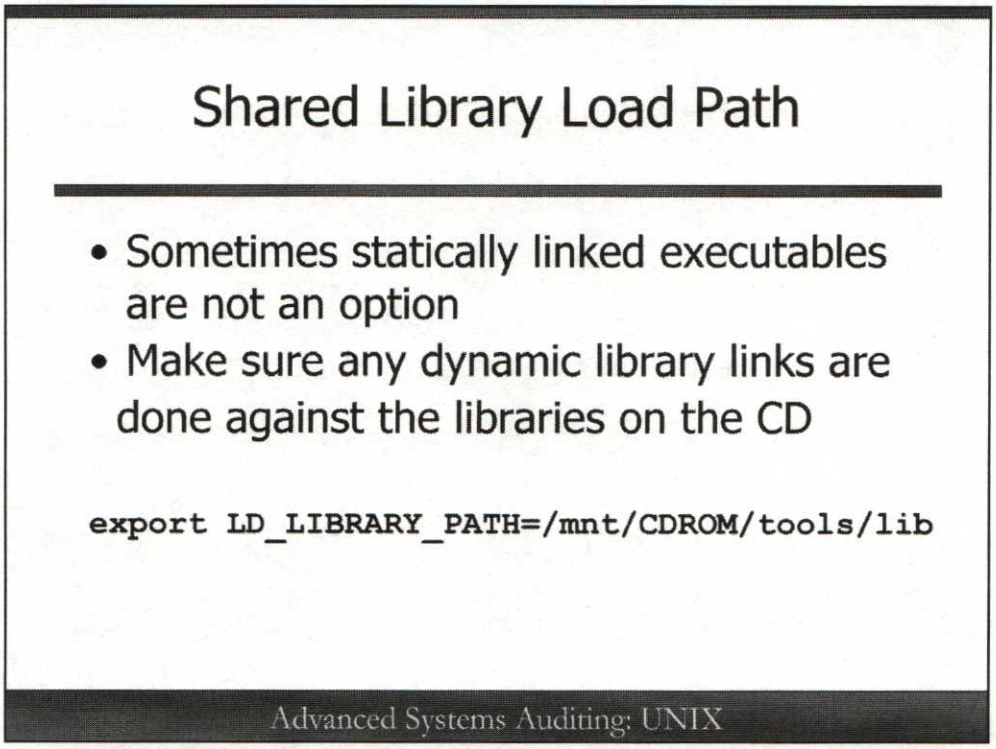
Dynamically linked executables are used often because they take less space. Many programs utilize the same basic system libraries for accessing the disk, writing to the screen, and so on. There is little need to have these stored in the program itself—the information is common to many programs. Although this makes great sense from a practical point of view, it is unfortunately another point of attack for system crackers. To avoid dependency on what will be suspect system libraries, we will compile our tools using gcc with the -static option so that the program that is generated is what is called a static binary or standalone executable. I've picked an example of a dynamically linked executable and am introducing the ldd command to visually illustrate my point.

Here's an example of dependency discovery using the ldd command:

```
# ldd /usr/bin/passwd
        linux-vdso.so.1 => (0x00007fff7fa7c000)
        libpam.so.0 => /lib/x86_64-linux-gnu/libpam.so.0 (0x00007f1de004c000)
        libpam_misc.so.0 => /lib/x86_64-linux-gnu/libpam_misc.so.0 (0x00007f1ddfe48000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f1ddfc28000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f1ddf868000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f1ddf664000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f1de0262000)
```

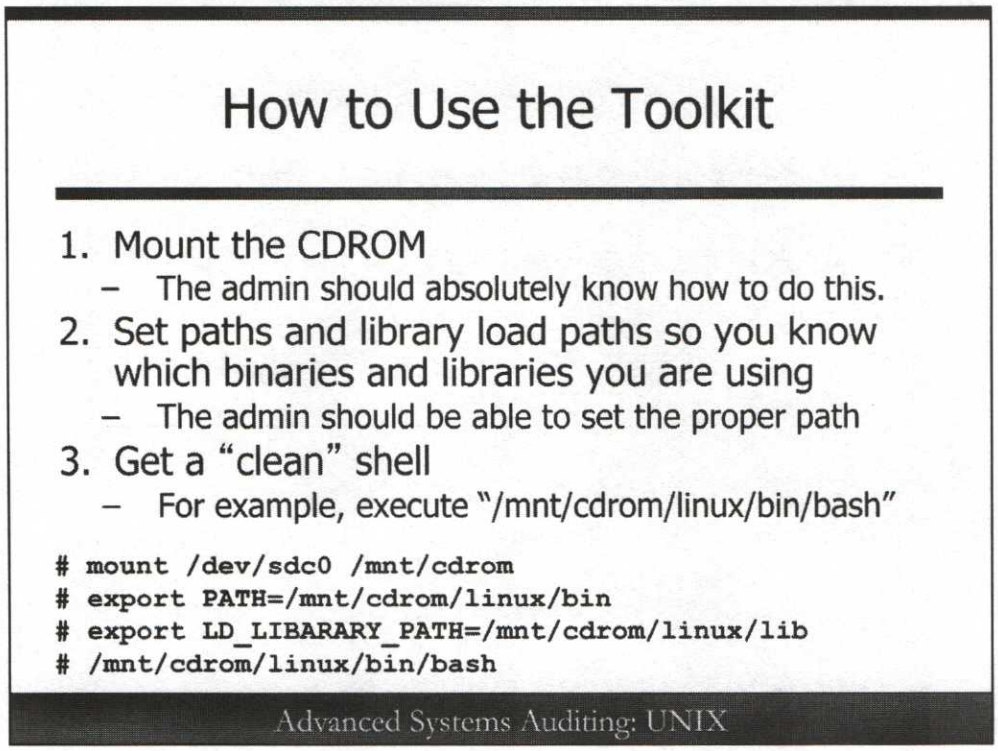
Notice that /usr/bin/passwd has dependencies that are not at all surprising—libpam (PAM=pluggable authentication modules), for instance—and the ever present libc and ld-linux for Linux-based systems. Should any of these libraries become corrupted or worse, compromised, it would affect the usage of /usr/bin/passwd and any other files that links against it.

© 2016 David Hoelzer & Enclave Forensics™



There will be cases when a statically linked executable is not available. Whether you can't get the source for a utility and are forced to download ready-made packages (Solaris and other vendor OSes don't normally come with compilers), or time does not allow it, you must go with a dynamically linked file. There is nothing wrong with this provided you know what libraries you are linking against. The special environment variable *LD_LIBRARY_PATH* defines in what order which directories are searched. This is where you specify the directory of the libraries you copy to CD when the time comes for you to use these utilities. Copying the /lib and /usr/lib (from a live OS file system CD if possible) of Linux and many other operating systems (in fact, several OSes have /lib --> /usr/lib anyway) will take care of all of the system programs. /usr/local/lib can be included from an installed machine if you know it's clean and certain libraries could be needed. It is important to note that setting LD_LIBRARY_PATH adds to the system library search path, specifically in front of the existing path. It is important to know that you are superseding the order with your CDROM-provided directories, not replacing the path entirely.

© 2016 David Hoelzer & Enclave Forensics™



With our tools/audit CD built, we are now ready to work with the administrator. The general outline of commands is listed on the slide. Let's work through them.

First, the administrator mounts the CD. Where will it be mounted? How should it be mounted on a specific system? These are questions that the administrator himself should be able to answer because it's his system! Once the disk is mounted, however, the very first things that you would like to do are to reset two environment variables. The first one is the PATH variable. This is used by the shell to determine the search path for binaries. Note that we are setting this to be the path to the binaries from our disk.

The second variable that is set is the dynamic library load path. If we fail to set this variable, then the commands that we run that are dynamically linked will continue to use binaries from the system being inspected. Setting this forces the system to use our prepared libraries from a known good system.

With those two variables set, the final step is to launch a clean shell from the CD. We're now ready to go!

© 2016 David Hoelzer & Enclave Forensics™

Command: `script`

- It is vitally important to take detailed notes
- Can fill in blanks later when we're at our office
- The **`script`** command takes notes for us, making a typescript of everything printed on the terminal. Usage:

```
script audit_output.txt
```

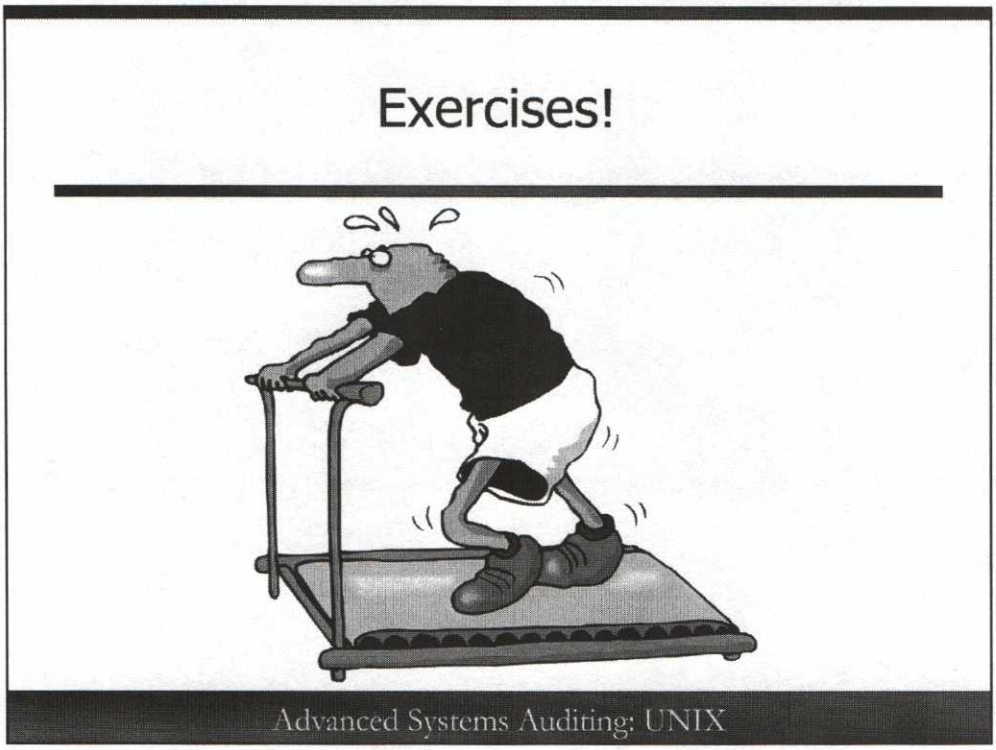
Advanced Systems Auditing: UNIX

Once the fresh shell is running, a fantastic tool to have the administrator run is the 'script' tool. In fact, even if you don't choose to build an audit CD as we're recommending, we'd still strongly recommend that the audit always begin with this command.

When you run this command, it will begin to record every command that is typed and everything that appears in the terminal window where it's running. When you're finished with all of the tests, simply have the administrator type the word "exit" or press "Control-D." This will cause the recording to stop and generate a file. Within the file, you will have a starting timestamp, a record of everything that was done, a record of all of the results, and an ending timestamp.

This should definitely become a part of your audit evidence. Not only will it document for you everything that was done, but it can also be a very valuable piece of evidence should someone eventually claim that you did something that you didn't!

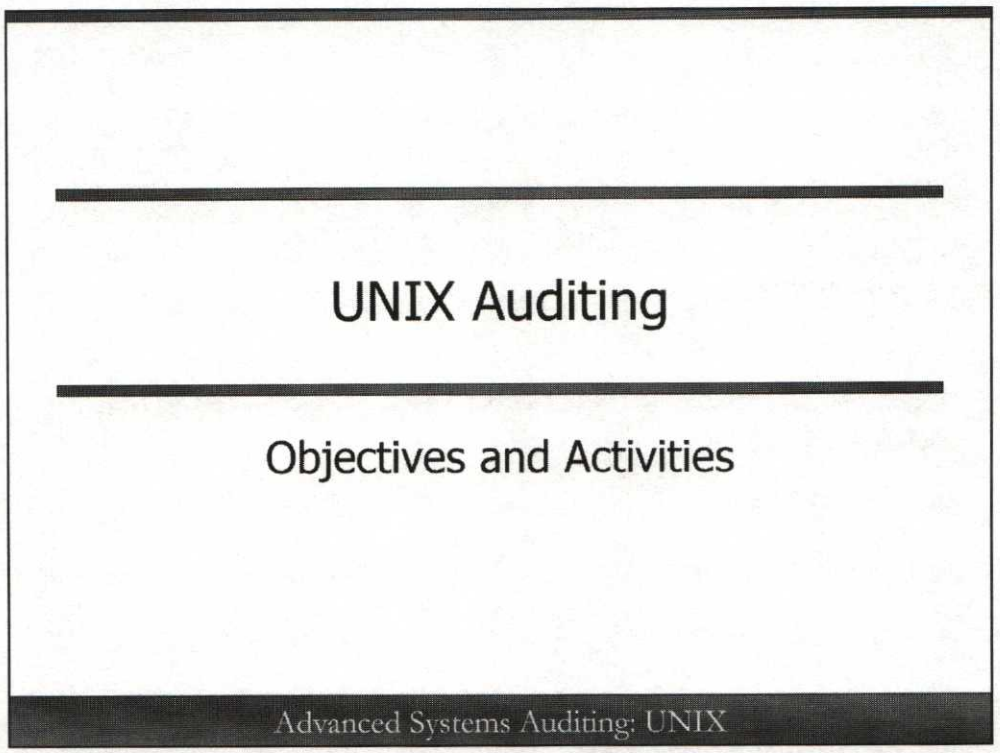
© 2016 David Hoelzer & Enclave Forensics™



The exercises that go with today's material are intended to be used to broaden your knowledge and at the same time, work you through a baseline audit of a UNIX system. To this end, we recommend that you open up your workbook to the Day 6 exercises and begin leafing through until you see something that looks either interesting or unfamiliar. If you are not particularly familiar with UNIX, then we recommend that you start with the very first exercise in Day 6.

We also recommend that throughout the day, you take the opportunity to try things as we go. If you see something interesting happening on the screen or in the slides, try it out!

© 2016 David Hoelzer & Enclave Forensics™



This page intentionally left blank.

© 2016 David Hoelzer & Enclave Forensics™

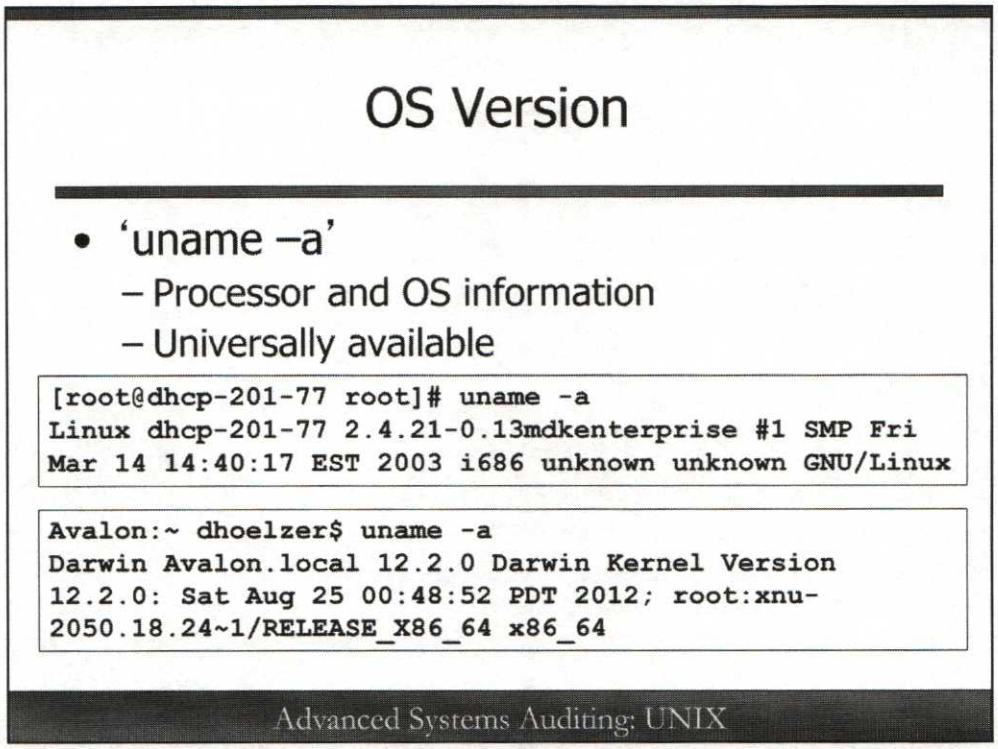
Audit Objective

- Objective: System Information
  - Identify system type
  - Identify patch level
  - General system information
- Audit Activities:
  - Uname
  - Patchdiag, etc.

Advanced Systems Auditing: UNIX

Now that we can build a CD, let's look at specific audit objectives and activities. We begin here with the examination of system information including the system type, patch level, kernel revision, and so on.

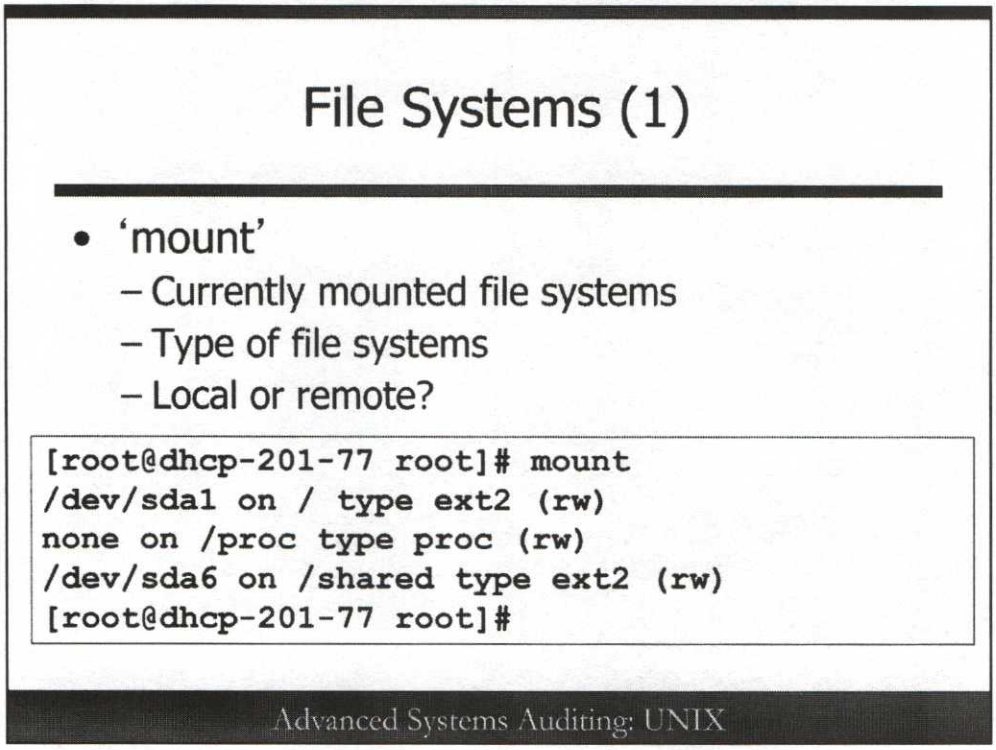
© 2016 David Hoelzer & Enclave Forensics™



When an attacker compromises or attempts to compromise your system, one of the very first things that he or she will do is attempt to run the 'uname' tool. This tool reveals information about the kernel, the processor on the system, build time, and so on. Especially in a reconnaissance phase, this tool is invaluable for an attacker if he can run it because it allows him to narrow the field to exploits that are likely to succeed against the architecture in question.

For an auditor, the tool is useful because it allows the auditor to gather baseline information about the machine.

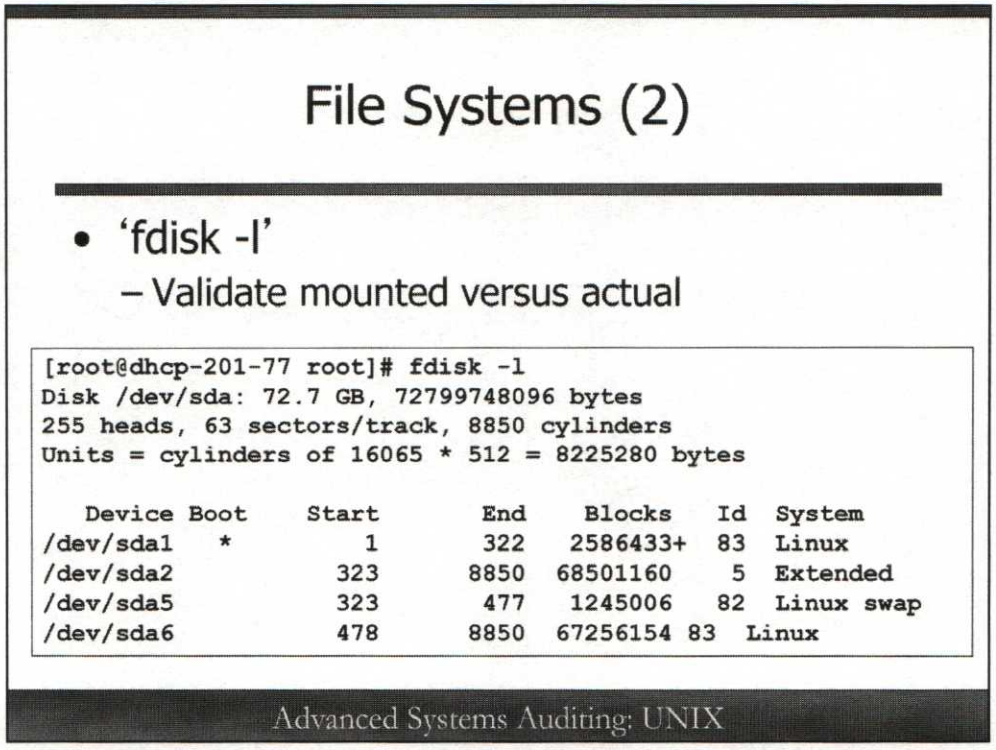
© 2016 David Hoelzer & Enclave Forensics™



In the process of creating the baseline, remember that we want to gather as much static information about the system as possible. One thing that tends to be fairly static is how the file systems are mounted. Although the file systems themselves might change over time, the configuration of the file systems is unlikely to change unless a new drive is installed or the system is rebuilt. The 'mount' tool can be used to gather this information. Without any arguments, it will describe all mounted file systems, permissions, and devices that are currently in use.

One of the things that we'd like to identify are file shares. We'll cover this more specifically later today, but because we're already looking at the mount command, it's worth mentioning now. If we are mounting remote file systems, or file shares, we can typically tell very easily. The identifying mark is some combination of IP address, host names, colons, and hostnames. These stand in stark contrast to local devices, which will typically be mounting something found in "/dev" or "/devices" or something similarly obvious.

© 2016 David Hoelzer & Enclave Forensics™

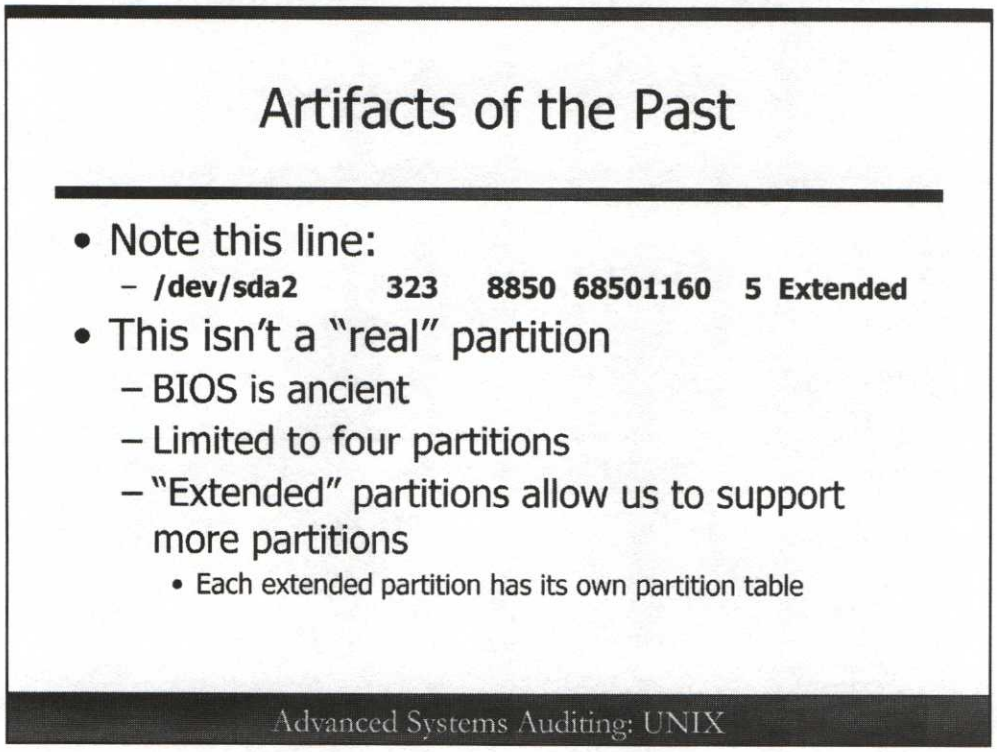


Although 'mount' can be used to show you what is mounted, is it possible that there is some portion of the drive that has not been mounted? Absolutely! In fact, this is a great way to create some "hidden" disk space. The space isn't really hidden; it's simply not mounted or available, so most administrators and auditors will easily overlook it. To see whether there are any unmounted or unusual partitions, the 'fdisk' tool can be used.

'fdisk' is intended to be used to manipulate the partition table of a hard disk, so caution is recommended in the use of the tool. It is very safe, however, to run 'fdisk' with an '-l' option, which will produce a list of all the partitions on the specified device or, if no device is specified, it will produce a list of all the partitions on all of the installed drives!

This is a tool where you can definitely find some variety. For example, on BSD-derived systems, including OS X, there is no '-l' option. There is a '-d' option to dump the partition table, but the result isn't very readable. These systems will often allow you to view the partition table by simply typing 'fdisk /dev/disk0' with no other options. Clearly, some research will be required to determine how this tool will work on a specific system because this same command will actually open the partition table for editing on other UNIX systems.

© 2016 David Hoelzer & Enclave Forensics™



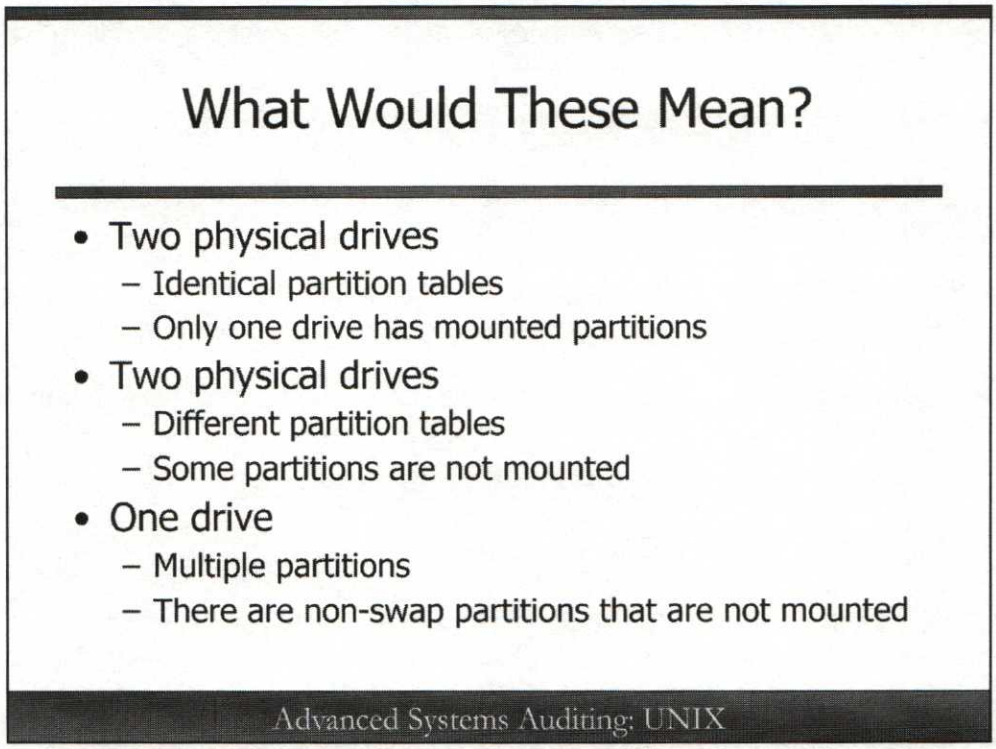
When looking at the partition information displayed by fdisk, we want to make sure that we know what sorts of things we are looking for. This also requires that we know which things we can safely ignore. For example, if there is a partition marked as "swap," you would want to be sure that it is actually being used, but otherwise it can be ignored.

A partition marked as "Extended," however, requires some explanation. Generally, you can ignore it. An extended partition is simply a method of efficiently allocating space on a hard drive. Originally, when the BIOS specifications were first created, no one ever thought that there would be a need for more than four partitions. After all, fixed disks weren't larger than 10 megabytes!

Backwards compatibility has lead to that original BIOS standard for drive partitioning remaining in use for many years, even today. To allow us to more effectively use the space and to support more than four partitions, it is possible to define an "Extended" partition. This is really just a pointer to another part of the hard drive. You will notice that an extended partition covers a large region of the drive and, quite likely, contains all of the partitions that appear after it in the list. This pointer tells the OS where it can find *another* partition table, allowing us to create as many partitions as necessary.

So, although an extended partition is very important and valuable, it can largely be ignored from our perspective. We are interested in the actual logical partitions that are being used to hold data!

© 2016 David Hoelzer & Enclave Forensics™



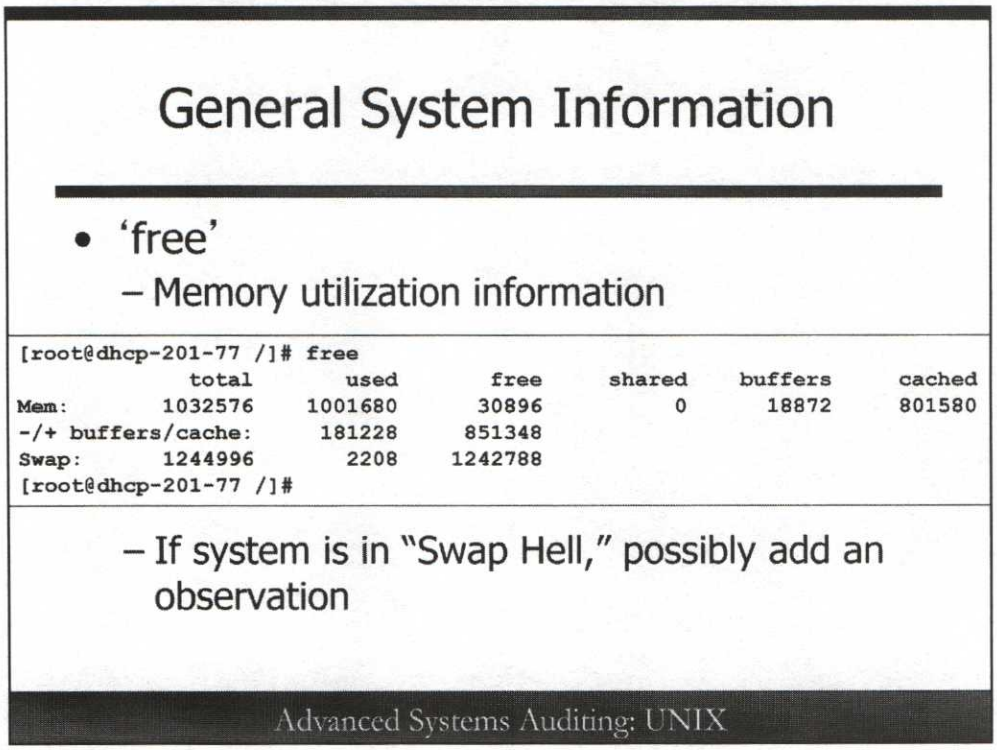
Consider the scenarios listed in the slide. These scenarios are listed, more or less, in the order of likelihood.

What would it mean if there were two physical drives in the system with identical partition tables but only one of them had mounted volumes? Quite likely, you are looking at a system with a mirrored drive (RAID 1). Of course, we should confirm this with the administrator.

On the other hand, if the drives have different partition layouts, they are independent drives. If all of the partitions are mounted or otherwise accounted for, there isn't any need for concern. However, if there are partitions that are not mounted, we should certainly inquire. It is possible that one or more of those partitions are related to recovery (OS X systems, for example) or possibly are used and managed by a database system (Oracle, MySQL, and PostgreSQL, for example, allow you to define a partition for exclusive use by the database).

Finally, if you find a single drive with multiple partitions and some of those partitions are not mounted or otherwise accounted for, you again would need to inquire. You could, again, be looking at recovery partitions or database partitions. Whatever the case might be, you need to determine precisely why the system is configured in that way.

© 2016 David Hoelzer & Enclave Forensics™

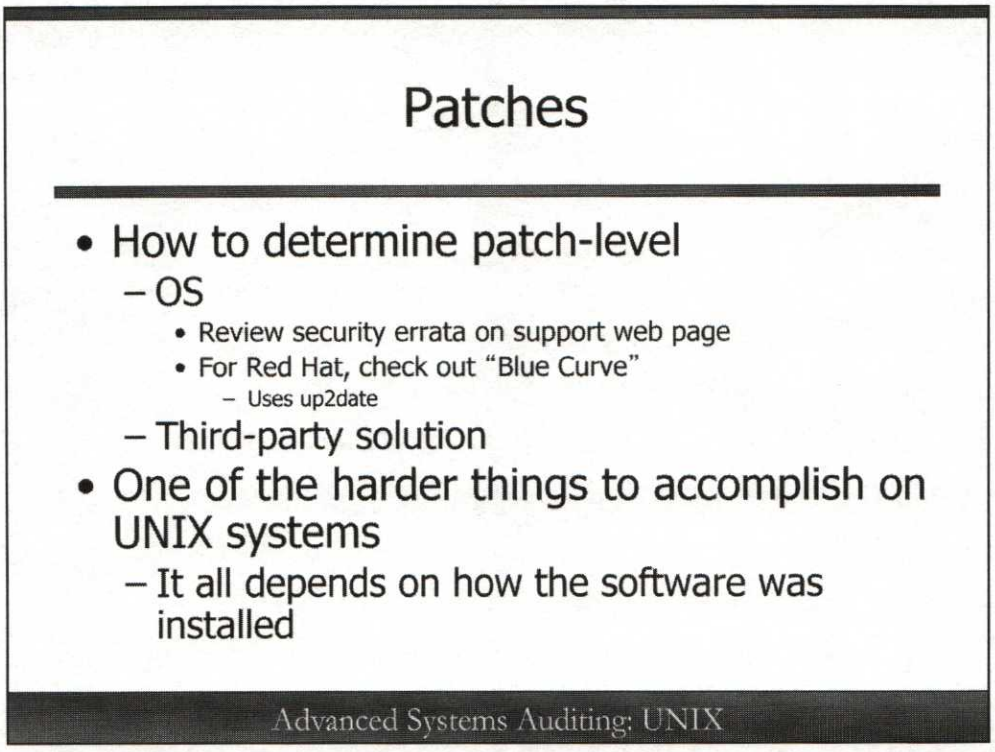


Another informative tool is the 'free' tool. This will allow us to identify several items for our audit. First, we can see how much physical memory is installed in the system. Second, we can see how big the swap partition is. Third, we can see how much space is currently in use and how much swap space is in use.

Though we are most likely concerned with the security of the machine, there are some interesting things to note when we examine these numbers. If we discover that there is a small amount of swap space in use (normal) and lots of free physical memory, everything is probably functioning normally. If, however, we discover that there is a great deal of memory in use, we could compare it to the process list and see whether the memory utilization reconciles to the actual usage. A large discrepancy could be an indication of covert activity. If we find that these numbers reconcile, however, and nearly all physical memory is in use plus a large amount of swap space, we might want to determine whether this is the typical load for the machine. If it is, we have found an issue, but not necessarily a security issue. The system has likely entered what is affectionately called "Swap Hell." The system is running at nothing near peak efficiency because most of the time is being spent swapping things on and off the disk!

If we find that there is a large amount of swap in use and little free memory, take note of that and come back later. Have the administrator run the 'free' command when we are finishing and see whether this situation has changed. If it hasn't, take the time to inquire. If this is typical, it would be really good to add an observation recommending that more RAM be purchased because this is definitely affecting the efficiency of the system.

© 2016 David Hoelzer & Enclave Forensics™



Examining the patch status of UNIX systems has become easier and easier over the years. Not so long ago, it was a long and tedious process to determine whether or not a system had been patched properly. These days, each UNIX vendor has its own proprietary solution for examining the patch status of the system.

Examples of these systems are tools like 'patchdiag' from Oracle, which will report the current patch status of a Solaris machine. Similarly, the program 'up2date' from Red Hat is used to track similar information for Red Hat. In fact, many of the Linux vendors are making available centralized update services akin to the Windows automated updates! For more information on this, check out Red Hat's "Blue Curve" desktop. Although they speak of it mainly as the consistent appearance, it also incorporates its patch-checking software.

© 2016 David Hoelzer & Enclave Forensics™

The two most common types of UNIX to come across in an enterprise these days are Linux (in many shapes and sizes) and Solaris. Although there are an enormous number of different Linux variants out there, most of them fall into two major groups: those that are derived from Debian and those that are derived from Red Hat.

If your organization is using Red Hat Enterprise, you might want to look into Satellite server. This system allows you to push out patches, centrally manage users via Kerberos, and manage general settings from a central console. In terms of patching, this is somewhat manageable as long as all of your software was installed using RPMs via Satellite.

For Debian-derived systems the primary tool that can easily be used to check on patch status is Aptitude. Unfortunately, there's nothing here to centrally manage what happens, and you do have to rely on the repository that your systems are pointed at to be up to date, but for software installed using the apt toolkit it does a nice job.

In the Sun/Oracle world, the first thing that should happen is that an administrator should subscribe to the SunSolve alerts system. There is a handy tool called "Patch Check Advanced" written by Martin Paul available that can help. This free tool for your modern Solaris systems can be obtained from http://www.par.univie.ac.at/solaris/pca/.

AIX is one of the more difficult systems for patch management. The oslevel command will give you information about the "Maintenance Level" of the system. Adding the '-s' option will give you service pack information as well.

© 2016 David Hoelzer & Enclave Forensics™

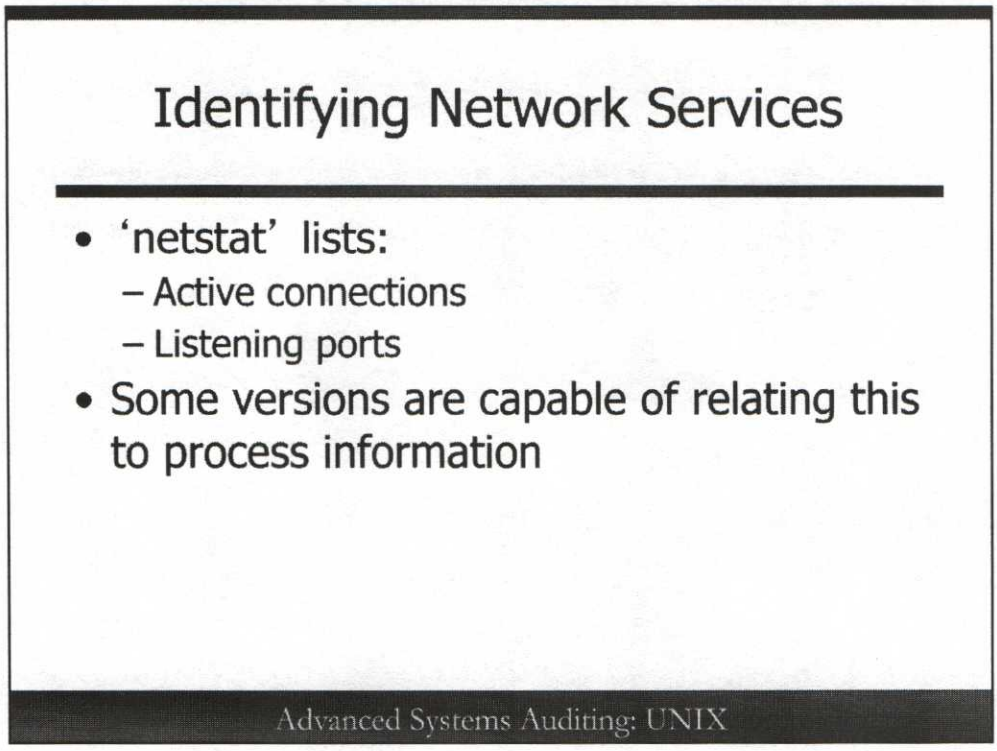
Audit Objective

- Objective: Operating Profile
  - Identify Network Services
  - Identify Local Services
  - Identify Network Behavior
- Audit Activities:
  - Netstat
  - Lsof
  - Ps
  - top

Advanced Systems Auditing: UNIX

Our next area of interest is the operational profile of the system. We are interested in identifying all of the local and network services that are running and available on the system so that we can evaluate the overall security profile of the system, particularly on the network level. We have a number of tools that can assist us here, every one of which we would want to include on a tools CD.

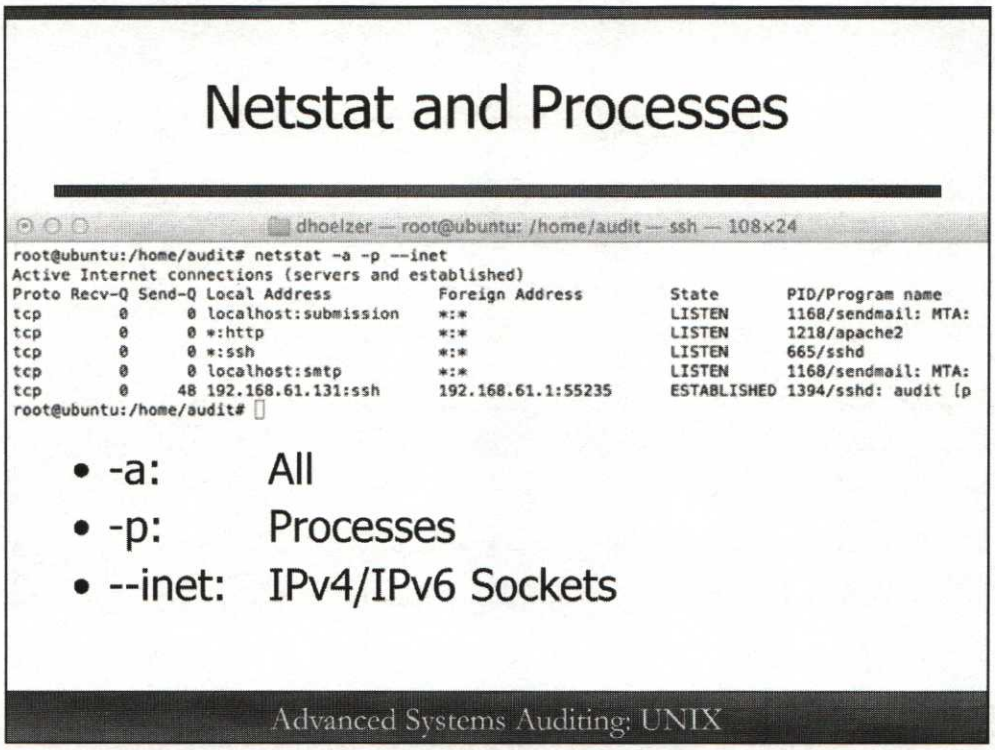
© 2016 David Hoelzer & Enclave Forensics™



Netstat lists all active connections in addition to the ports where programs are listening for connections. Simply use the command netstat -a -p --inet for a listing of this information. It is important to note that not all UNIX versions support the '-p' option for netstat, so we might need to use another tool to find the process information. We will provide a solution to that problem shortly. In the example on the next slide, a partial netstat listing is shown.

Note that I've used some special arguments. The -a argument makes netstat display the entries for all sockets, including those that are simply listening. This is useful for identifying all of the ports that are open. Using this information, the system administrator can prune system services, eliminating those that aren't absolutely essential.

© 2016 David Hoelzer & Enclave Forensics™

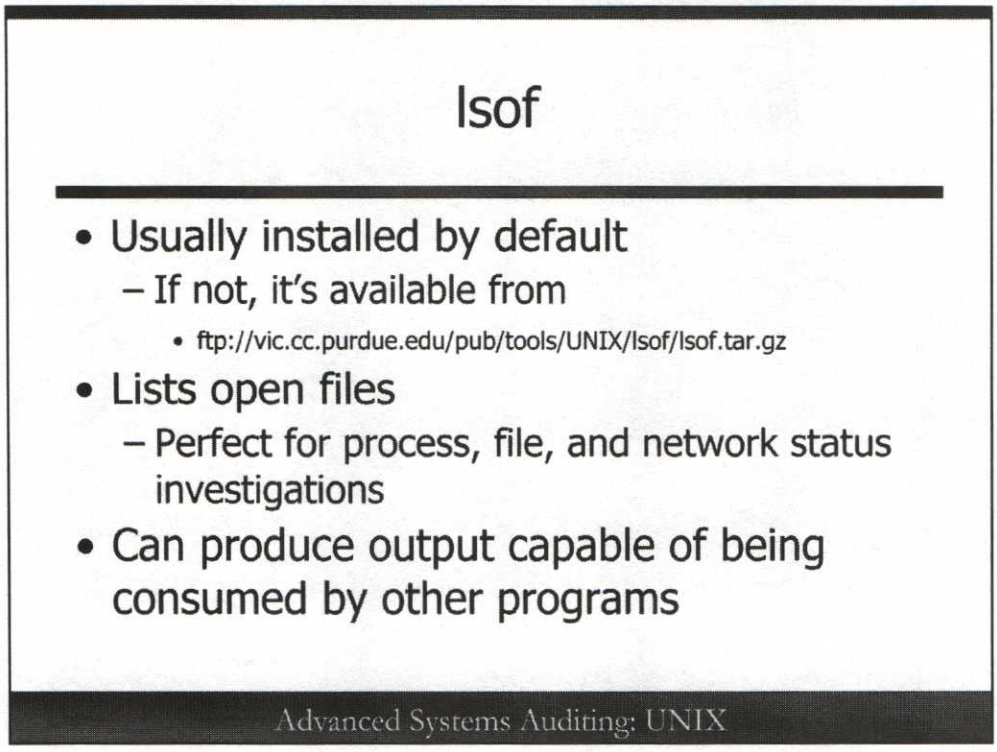


Another reason to run netstat is to see whether any unknown server programs are actively listening for connections. In this example, one can plainly see that there is an ssh session established with a remote-host – 192.168.61.1. All of the other services that our machine is serving can be seen too.

Netstat is a fairly standard tool to find on most operating systems these days. Even though it's known as a UNIX tool, it is actually more a part of the IPv4 and IPv6 implementation of Berkeley sockets. For this reason, you'll not only find it on POSIX-compliant systems, but you will also find it on mainframe operating systems, legacy mini-computers, and more.

The '-p' option is the primary point for this slide. You can see that the process holding open each port is identified for us. Although the '-p' option is extremely valuable, it won't always be available! For example, you will find that OS X and other similar systems lack the option completely! What can be done in a case like this to map network ports to running processes?

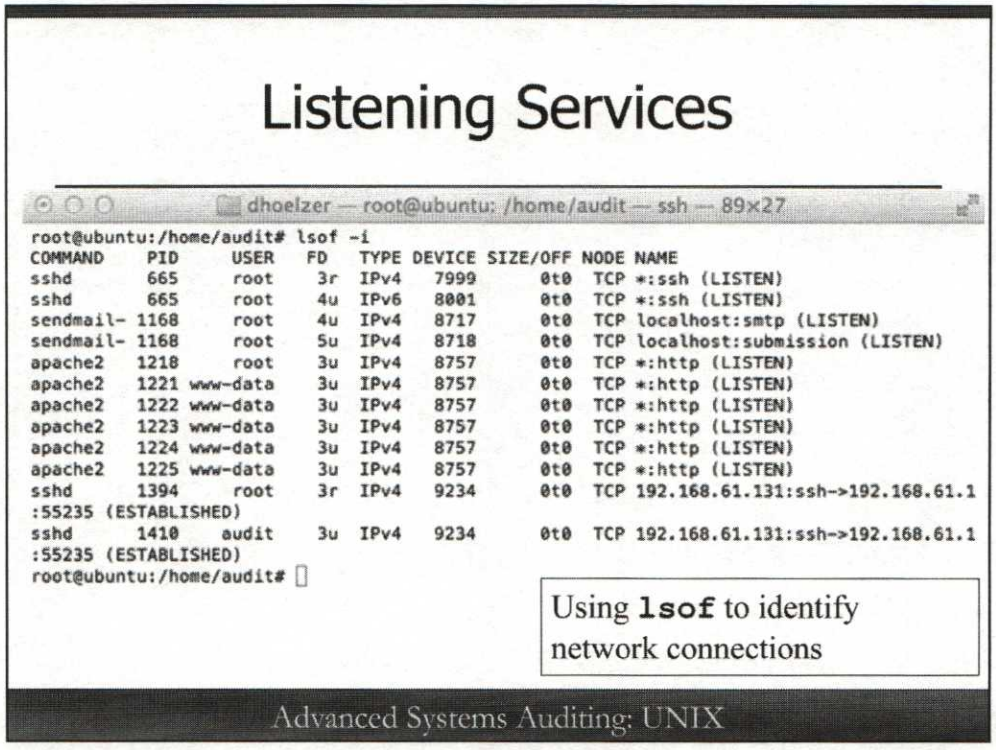
© 2016 David Hoelzer & Enclave Forensics™



Lsof to the rescue! Because everything in UNIX is a file, including network ports and connections, and because lsof will list open files, this tool can be used to list all files that are network ports and that are currently open!

This tool has become a part of the standard base for UNIX these days. Even so, you will find some systems that do not have the lsof tool. If you happen to run across one, you can still obtain the source code for lsof from Purdue university free. It can then be compiled for the specific platform that you're dealing with and added to your audit CD.

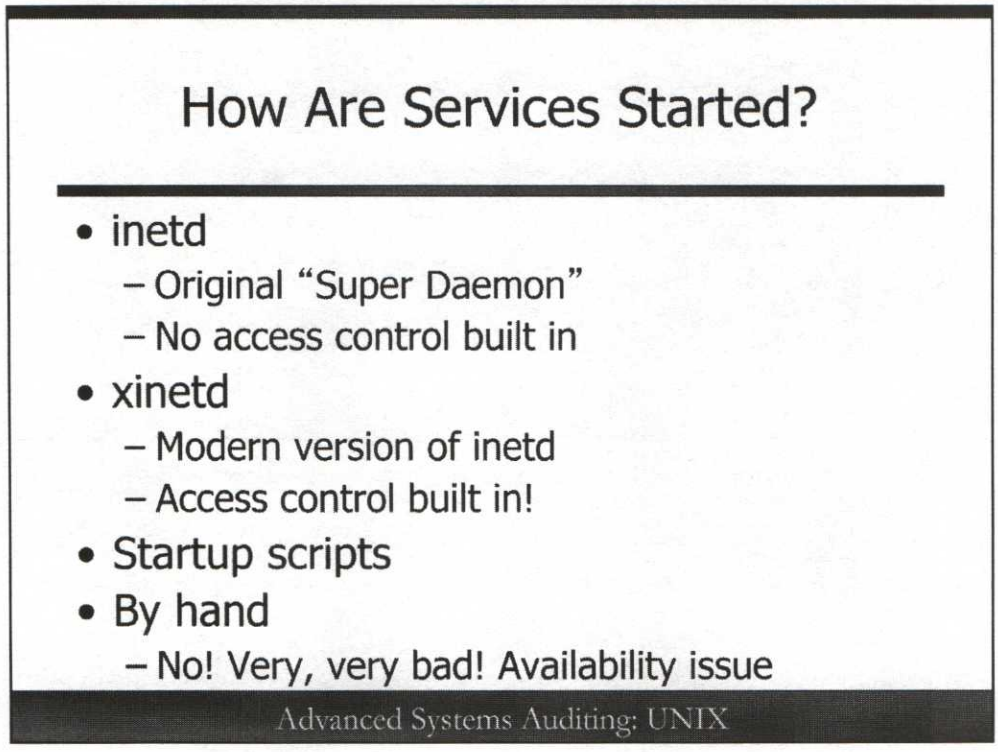
© 2016 David Hoelzer & Enclave Forensics™



As you can see, lsof -i displays a listing that is similar to that of the netstat command discussed earlier.

Although there are certain columns, like the PID, that will change, much of this information should be a part of a baseline. In fact, you likely have in mind that the administrator should be very interested in using our baseline script to perform periodic, automatic analysis of his system in order to have an early warning of unauthorized operational changes.

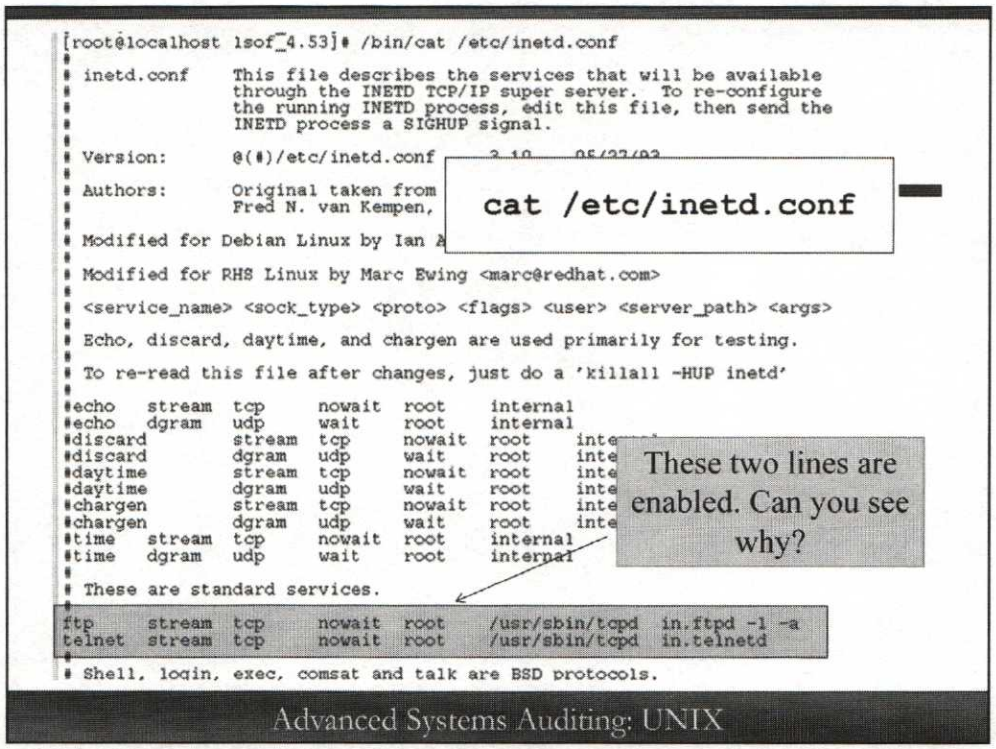
© 2016 David Hoelzer & Enclave Forensics™



Network services on a UNIX machine can be started in a number of different ways. One of the most common models is to run the service through another service, known as a "Super Daemon." A daemon on a UNIX machine is a process that starts and then continues to run with no user interaction, periodically taking actions or waiting for a network connection to act. An example is something like the mail daemon. The mailer starts up and then sits waiting patiently for a mail message to come along so that it has some work to do.

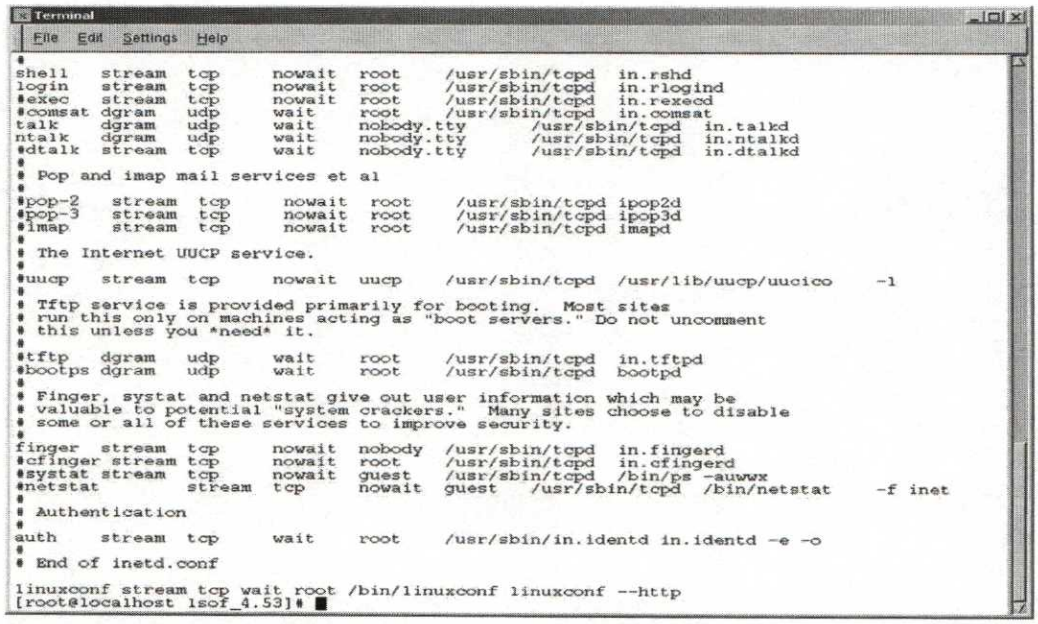
Starting a service by hand is always possible, but would be atypical for a server deployment. Relying on manually starting services would be extremely fragile because any system reboot would remove those services. As it turns out, this type of thing can go unnoticed because UNIX systems tend to have very large "uptimes." This means that, perhaps two years from now when there is a planned outage, the system will be restarted and a critical service will suddenly vanish. The worst part is that the people administering the system at that point will likely have no idea why the service has disappeared. ☺

© 2016 David Hoelzer & Enclave Forensics™



Below, you can see the rest of the file. Think of inetd as a service "broker." Requests for services are made through inetd, and the services that it brokers are defined in /etc/inetd.conf. One way to disable some of the extraneous services is to comment their entry out of the /etc/inetd.conf file. Here, it would be prudent to comment out pretty much everything but ftp (only because we want this to be an ftp server). Also, note in the next to the last column, the entry /usr/sbin/tcpd. This indicates that tcpwrappers is installed on this system. That's good information to have for later.

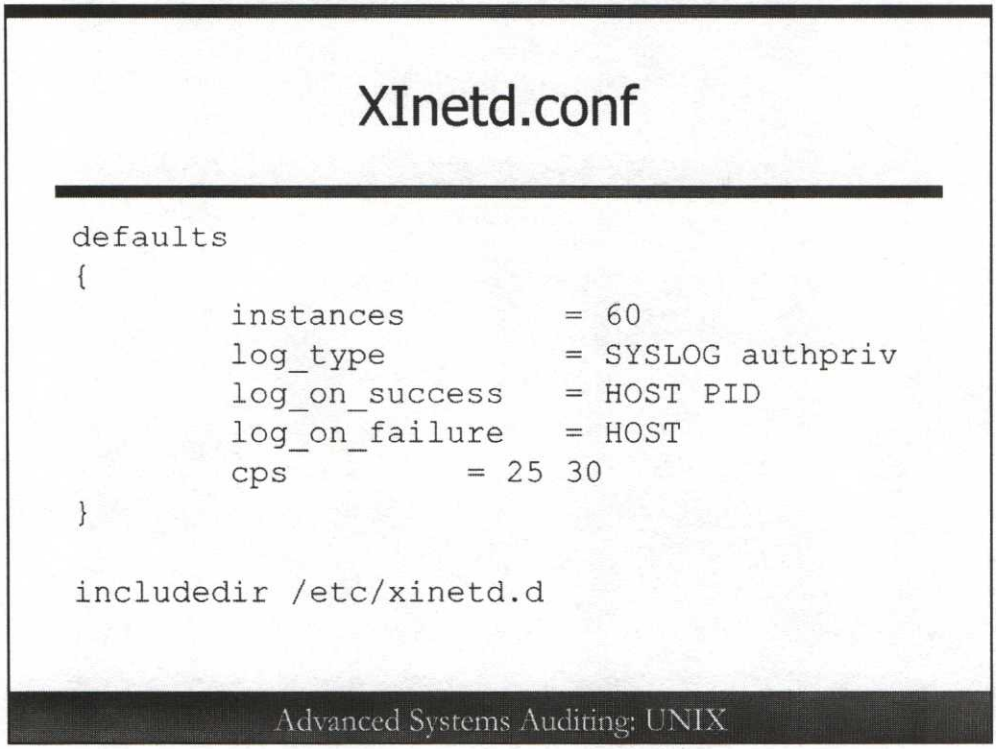
When looking at a system that is using inetd.conf, we should *always* expect to find that TCPWrappers has been used to protect services rather than having them running on their own with no wrapping.




```
#
shell   stream  tcp     nowait  root    /usr/sbin/tcpd  in.rshd
login   stream  tcp     nowait  root    /usr/sbin/tcpd  in.rlogind
#exec   stream  tcp     nowait  root    /usr/sbin/tcpd  in.rexecd
#comsat dgram   udp     wait    root    /usr/sbin/tcpd  in.comsat
talk    dgram   udp     wait    nobody.tty      /usr/sbin/tcpd  in.talkd
ntalk   dgram   udp     wait    nobody.tty      /usr/sbin/tcpd  in.ntalkd
#dtalk  stream  tcp     wait    nobody.tty      /usr/sbin/tcpd  in.dtalkd
#
# Pop and imap mail services et al
#
#pop-2   stream  tcp     nowait  root    /usr/sbin/tcpd ipop2d
#pop-3   stream  tcp     nowait  root    /usr/sbin/tcpd ipop3d
#imap    stream  tcp     nowait  root    /usr/sbin/tcpd imapd
#
# The Internet UUCP service.
#
#uucp    stream  tcp     nowait  uucp    /usr/sbin/tcpd  /usr/lib/uucp/uucico     -l
#
# Tftp service is provided primarily for booting.  Most sites
# run this only on machines acting as "boot servers." Do not uncomment
# this unless you *need* it.
#
#tftp    dgram   udp     wait    root    /usr/sbin/tcpd  in.tftpd
#bootps dgram   udp     wait    root    /usr/sbin/tcpd  bootpd
#
# Finger, systat and netstat give out user information which may be
# valuable to potential "system crackers."  Many sites choose to disable
# some or all of these services to improve security.
#
finger  stream  tcp     nowait  nobody  /usr/sbin/tcpd  in.fingerd
#cfinger stream tcp     nowait  root    /usr/sbin/tcpd  in.cfingerd
#systat stream  tcp     nowait  guest   /usr/sbin/tcpd  /bin/ps -auwwx
#netstat        stream  tcp     nowait  guest   /usr/sbin/tcpd  /bin/netstat     -f inet
#
# Authentication
#
auth    stream  tcp     wait    root    /usr/sbin/in.identd in.identd -e -o
#
# End of inetd.conf

linuxconf stream tcp wait root /bin/linuxconf linuxconf --http
[root@localhost lsof_4.53]#
```

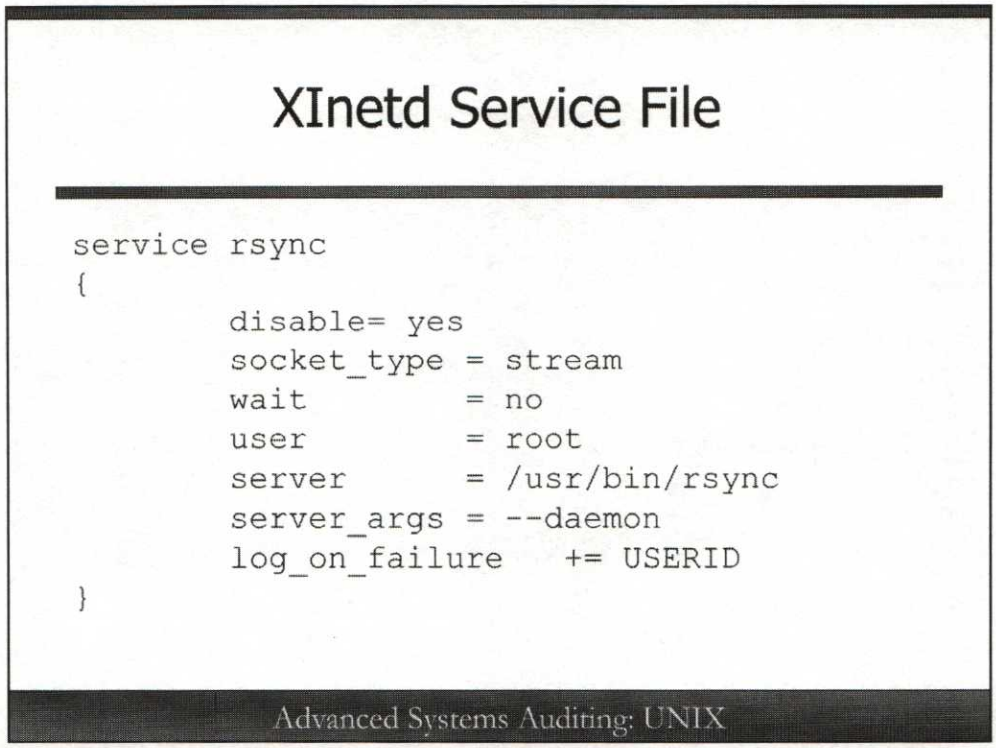
© 2016 David Hoelzer & Enclave Forensics™



A more modern version of inetd exists today, xinetd. The configuration of xinetd is quite different from inetd, relying on a special directory, '/etc/xinetd.d', which will typically include a configuration file for every service available on the system. A very important thing to know is that you cannot assume that just because a configuration file exists, the service must be running! It is quite typical for there to be an entry for many services that are marked "disable = yes"!

The main difference between xinetd and inetd is that xinetd incorporates TCPWrappers into xinetd itself! Now access control is provided from the super daemon without a need to call tcpd for each of the services as they start.

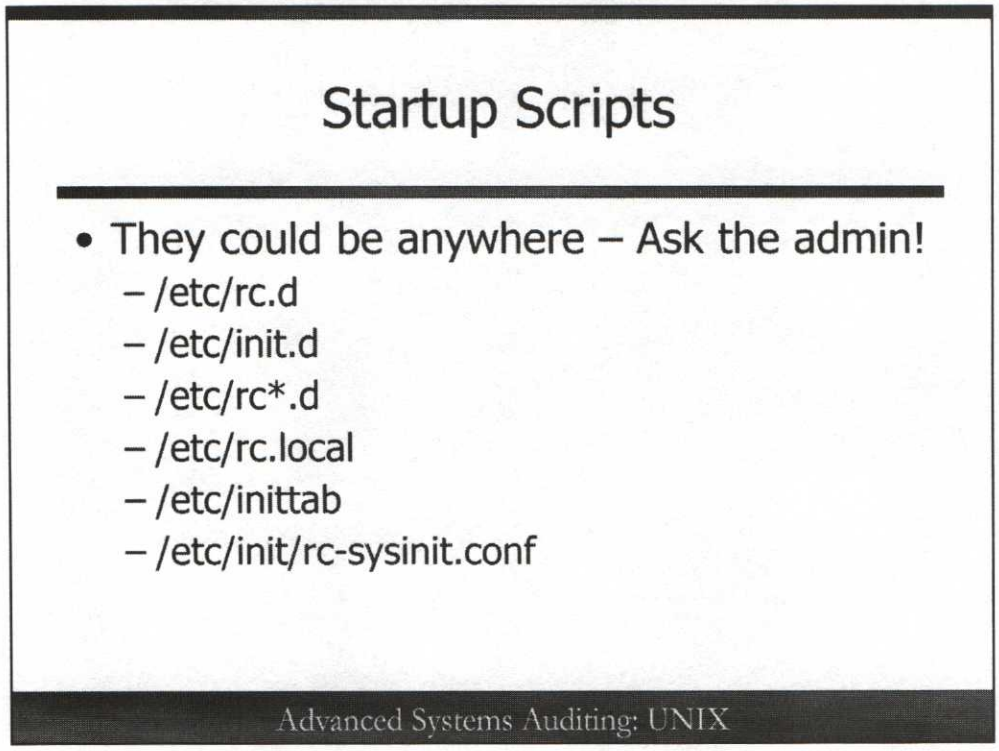
© 2016 David Hoelzer & Enclave Forensics™



Here is a sample of an /etc/xinetd.d/ script for rsync. The manual page describes what each of the various configuration options do. In this case, we will decipher them all for you:

Disable	Sets this service to "disabled" (not running).
Socket_type	Stream vs. DGram, which equate to TCP vs UDP.
Wait	How long before a new connection can come in?
User	User ID under which the service should be run.
Server	Location of the binary to run when there is a permitted connection.
Server_args	Command-line arguments to send to the server.
Log_on_failure	Although the /etc/xinetd.conf file specifies default logging options, it is possible to override these through log_on_failure and other configuration commands. In this case, notice that the argument is "+=USERID". This means that the user ID should be added to the default log options.

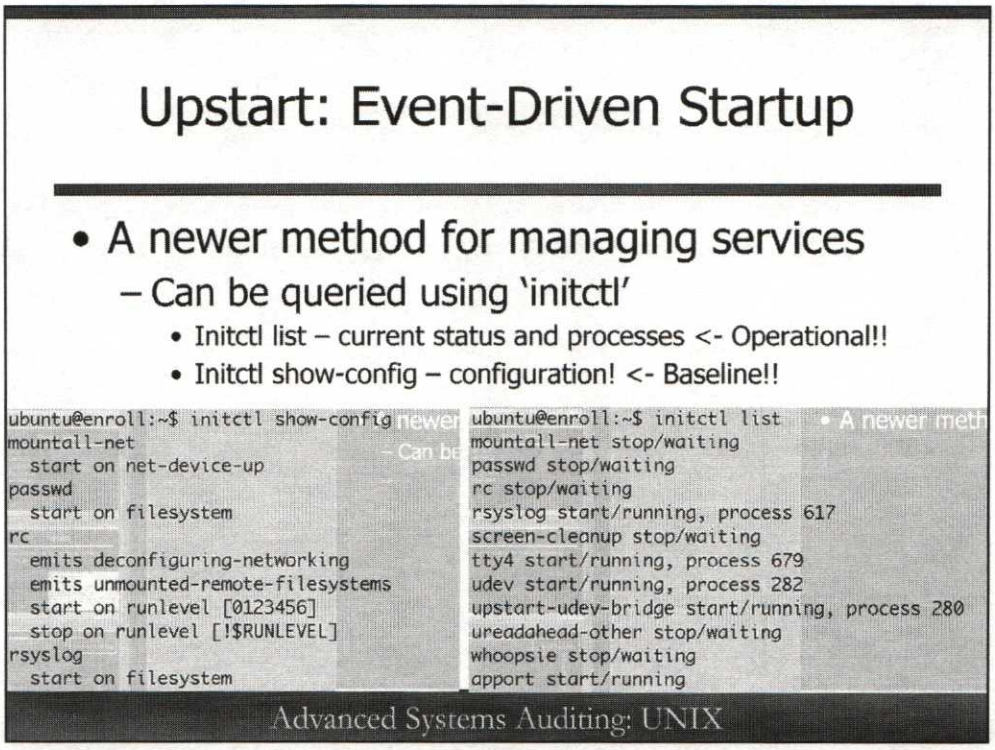
© 2016 David Hoelzer & Enclave Forensics™



The startup scripts for almost all systems will be found somewhere in the /etc directory. We can't be more specific than that, but as you can see from the slide above, there are a bunch of places that you might find them. To complicate things, newer UNIX operating systems like OS X are doing away with these files entirely, using what's known as the "Launch Daemon," launchd. We'll come to that in a moment.

For more traditional UNIX systems, however, this is all controlled through text files.

© 2016 David Hoelzer & Enclave Forensics™



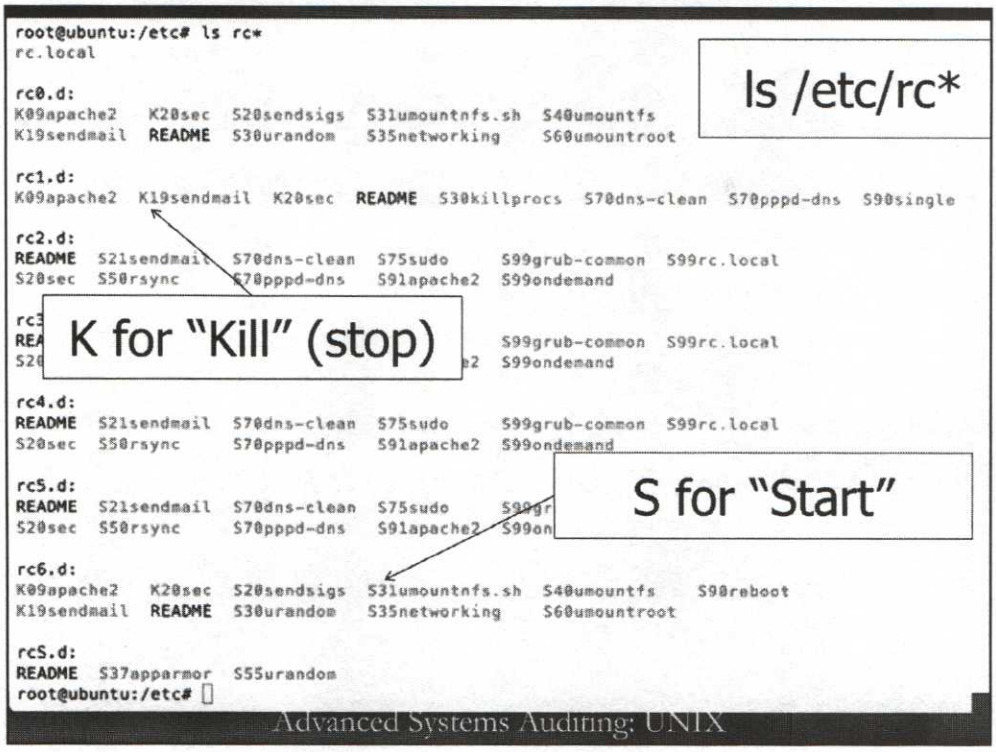
Some of our Linux systems these days are moving toward event-driven startup tools. Ubuntu, for instance, uses a package called "Upstart" for managing services. This is really just another example of startup scripts; however, digging through the configurations can be difficult if you don't realize that upstart is in use.

Event-driven startup utilities no longer require that everything in startup happen in the same order every single time. Instead, the startup process becomes multi-threaded. Each step in the startup process runs as soon as the prerequisites for it have been met. The main driver for this change is that people are very concerned with how quickly the login screen is presented to users these days. We are at the point where the login screen is now appearing in seconds, making the user feel like the system is super fast. In reality, though, all of the same startup still needs to happen; it's just happening behind the scenes while the login page is being displayed. This is why you might find that your system appears to be fully booted but is actually somewhat sluggish for the first few minutes after you log on.

To manage or view services with upstart, you must use the command 'initctl.' This command has its own help system built in, but the most common two commands that you will be interested in are 'show-config' and 'list.'

The 'initctl list' command provides a list of the current status of all of the installed services. This is very useful as an operational monitoring tool because it will show us everything that is running and the assigned process IDs. The 'initctl show-config' command, on the other hand, will show you the configuration status of all of the services regardless of whether they are currently running or not. This allows you to determine what *should* be running and should definitely be a key aspect of any baseline audit.

© 2016 David Hoelzer & Enclave Forensics™



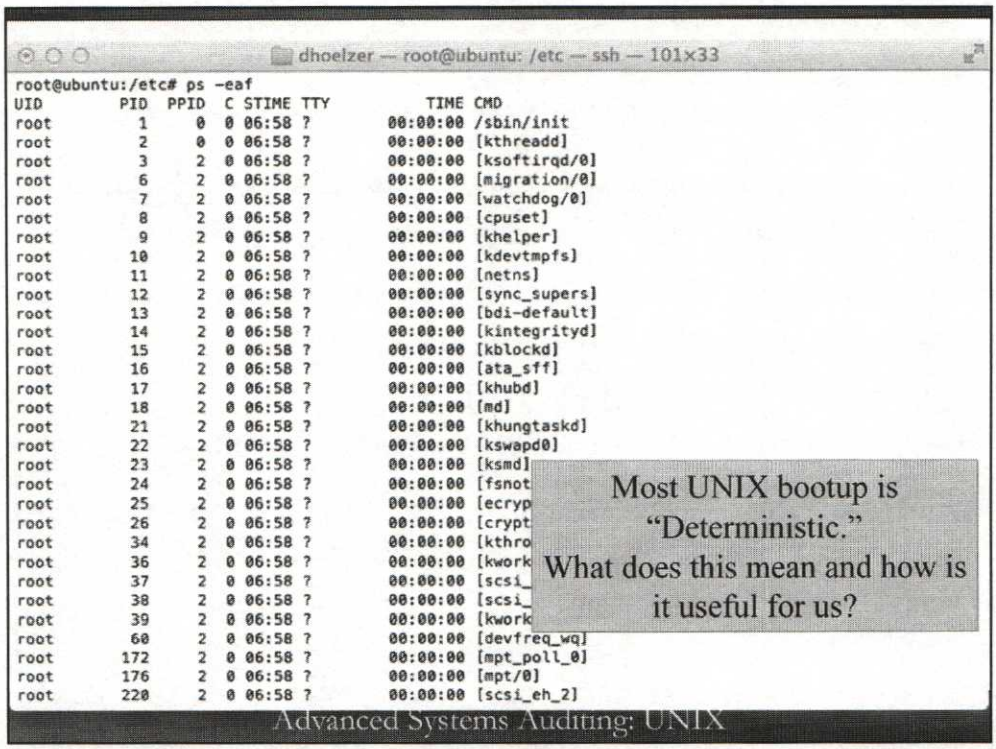
Here we are peeking under the hood of an Ubuntu system. From within the /etc directory, we are looking at the content of all of the "RC" directories. RC in this context stands for "Run Command."

Notice that each directory follows the pattern "rcX.d" where X is a number form zero through 6 or the letter S. These represent various "run levels" or levels of functionality. Unfortunately, the numbers themselves are quite arbitrary, though there are some conventions. For example, runlevel 0 typically represents shutting down. Runlevel 1 represents single user mode. Beyond this, however, you really will need to look at the documentation for the system in question to determine what a given runlevel means.

Notice that the files within the directory have two types of names. The vast majority begin with a capital letter "S" followed by a number and descriptive name, whereas a few begin with a capital letter "K." The way that this works is that the init (initialization) process will get a listing of all of the files in the directory for the matching runlevel. It then works through these files in order, which is the purpose of the numbering. There's nothing magical happening here; it's simply taking them in the natural sorted order for the files.

If a filename starts with an "S," then that script is run with the option "start". If the filename begins with a "K" (for kill), then the matching script is run with the option "stop." In this way, the administrator can granularly control precisely what will start and stop at the various runlevels.

© 2016 David Hoelzer & Enclave Forensics™



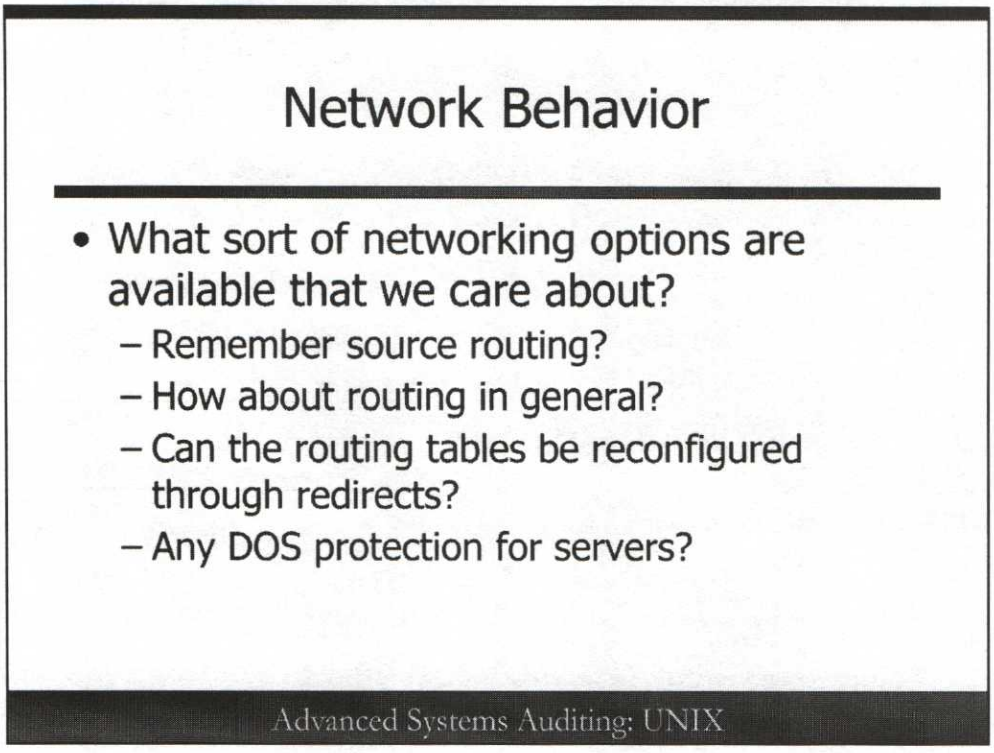
Although we have quite a few ways to check to see which services are running on our machine, let's not forget the ease and usefulness of our old friend "ps –eaf". "ps" lists the process table along with other pertinent information, including the owner of the process, the percentage of CPU and memory utilization, the terminal that the process is active on, and the start date of the process.

Keep an eye out for system processes (like syslogd and klogd) that have a start date that is later than some other processes. Such an occurrence could be evidence that the system has been compromised and the logging facility has been altered and restarted.

Another useful thing to note is that during the boot process of most UNIX systems, until network services finish loading, these UNIX systems are deterministic. This means that if you rebooted the system millions of times, it will perform the same tasks every single time in the same order until it starts speaking with the outside world and accepting input. Because we're talking about processes, let's apply it to process IDs. Every time a specific UNIX system starts, all of the boot-time processes will always have the same process IDs.

Why is this useful? Because it gives us another way to profile our system. Not only should we have a baseline of which services should be running, but we should also have the same PIDs for all of the boot-time services . If the PIDs have changed, something significant has changed on the system. The administrator might be able to explain the change. If not, consider asking for the system to be rebooted and examining whether or not the PIDs return to what is expected. If they do, something happened while it was running. If they don't, something has changed on the system that alters the startup pattern for the machine.

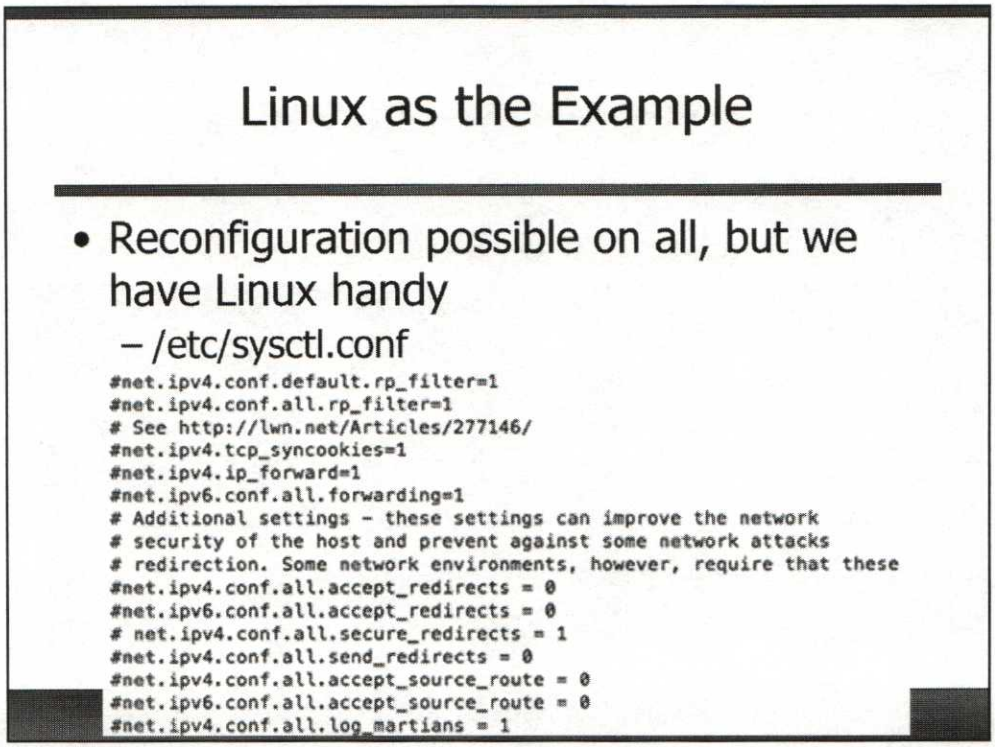
© 2016 David Hoelzer & Enclave Forensics™



Another aspect of our network-based operating profile is how our system reacts to stimuli aside from offered services. Most notably, we are interested in what sorts of routing behaviors might be configured and whether or not there are any protections against network-level attacks configured on the system. These could take the form of DOS attacks or source-routing attacks, convincing the device to route packets that it normally would not because it is likely not acting as a router, and so on.

Most UNIX systems have a fairly robust set of controls surrounding the network configuration that can be examined. Part of the reason for this is that UNIX systems have been in the role of network servers, firewalls, and routers for many years. As a result, many of these services have become a native part of the operating system.

© 2016 David Hoelzer & Enclave Forensics™

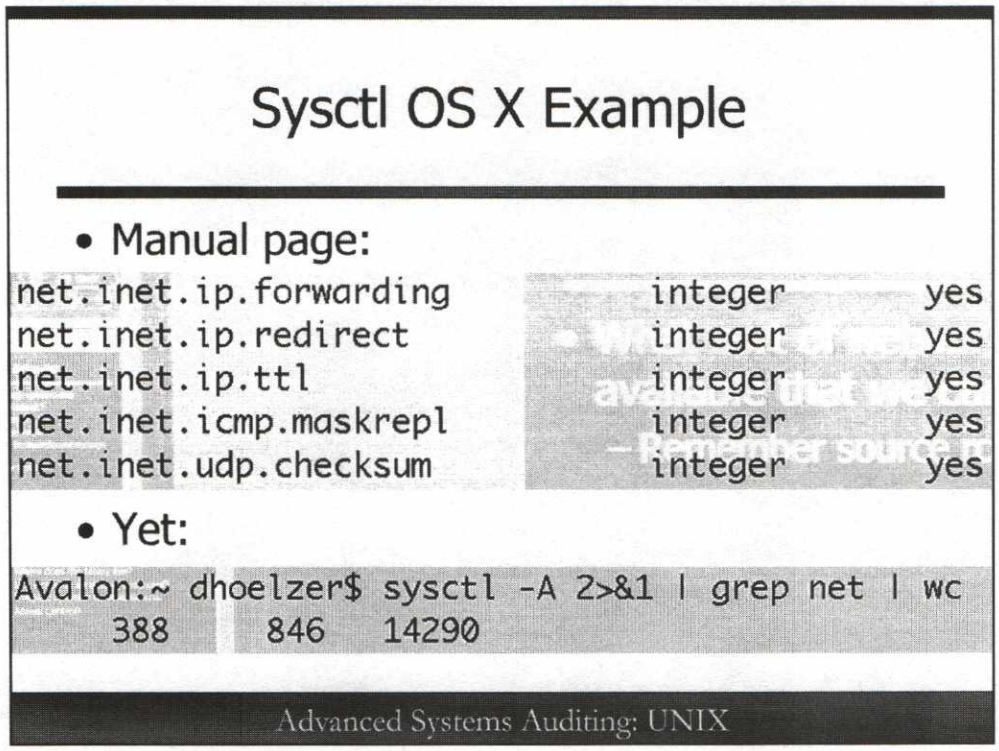


All of the major UNIX systems can be reconfigured in this regard, but how to do it or how to check it will vary by system. Let's look at Linux because it's so handy.

Many of the items that we are interested in are controlled through the sysctl (System Control) configuration. This file is typically found in /etc/sysctl.conf. You might not find all of the options listed in this slide in this file, but they can be put there if you are using a modern version of Linux. Take a look and see how many of these you can identify. Your instructor will take some time to explain some of them as well.

Could someone, such as a security analyst or a system administrator, determine which settings would be appropriate based on our organizational standards for network security? Could we script a test for these settings?

© 2016 David Hoelzer & Enclave Forensics™



As mentioned, what can be controlled with sysctl really does depend on the particular UNIX system. Although there are actually a large number of settings available, the 'man sysctl' manual page lists a significantly lower number. You can see this by comparing the manual page contents from OS X (in the slide) with the output of the 'sysctl –A' command on the same system. What's the story?

Well, as it turns out, manual pages are not always complete. I know that this might not surprise you. ☺ At the bottom of the slide, we are running the 'sysctl –A' command (which, by the way, is another great thing to baseline) to inspect the current configuration of all of the system parameters. Pulling out network-related items and using "word count" to count them up, we find that there are 388 items available, far more than we found in the manual page!

What's the lesson? Explore… and know your system! This goes for both us as auditors and for the administrators as well. It would be a rare find to have an administrator who is familiar with all 388 settings already, and those are just the network settings!!

© 2016 David Hoelzer & Enclave Forensics™

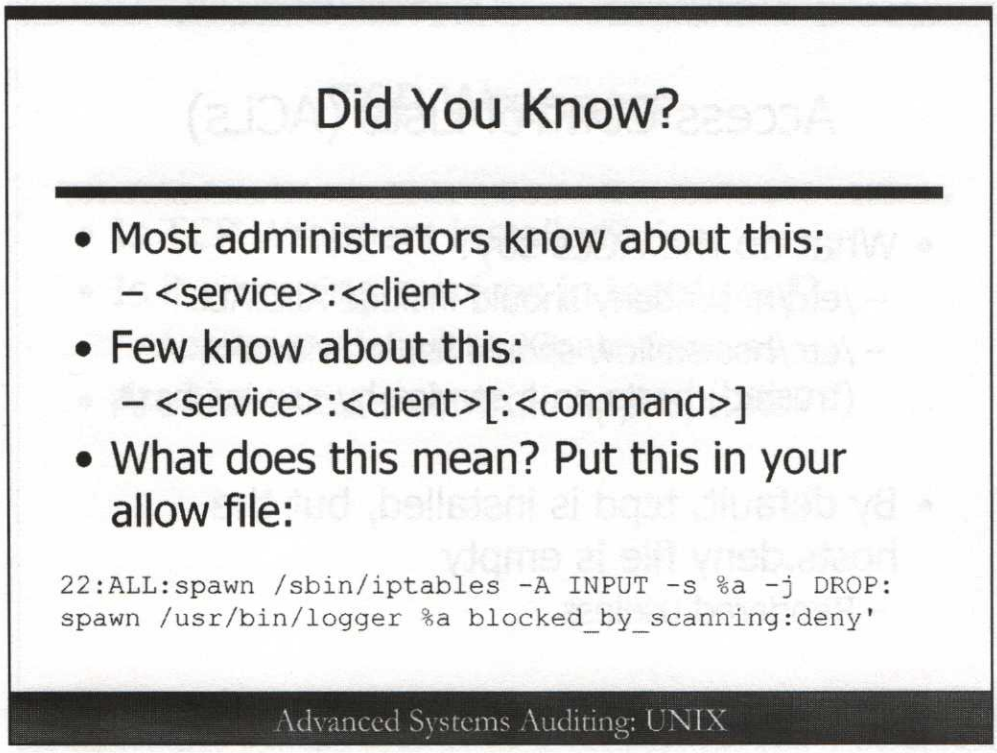
Audit Objective

- Objective: Unauthorized Access
  - Examine network-level access control
- Audit Activities:
  - TCP Wrappers
  - Hosts.allow
  - Hosts.deny

Advanced Systems Auditing: UNIX

Our operating profile took into account which services were running on the system, but now we need to examine the access controls surrounding those services that should be running. We discussed xinetd and TCP Wrappers a bit, but now we will be more focused on the rules that control who can connect to which services.

© 2016 David Hoelzer & Enclave Forensics™

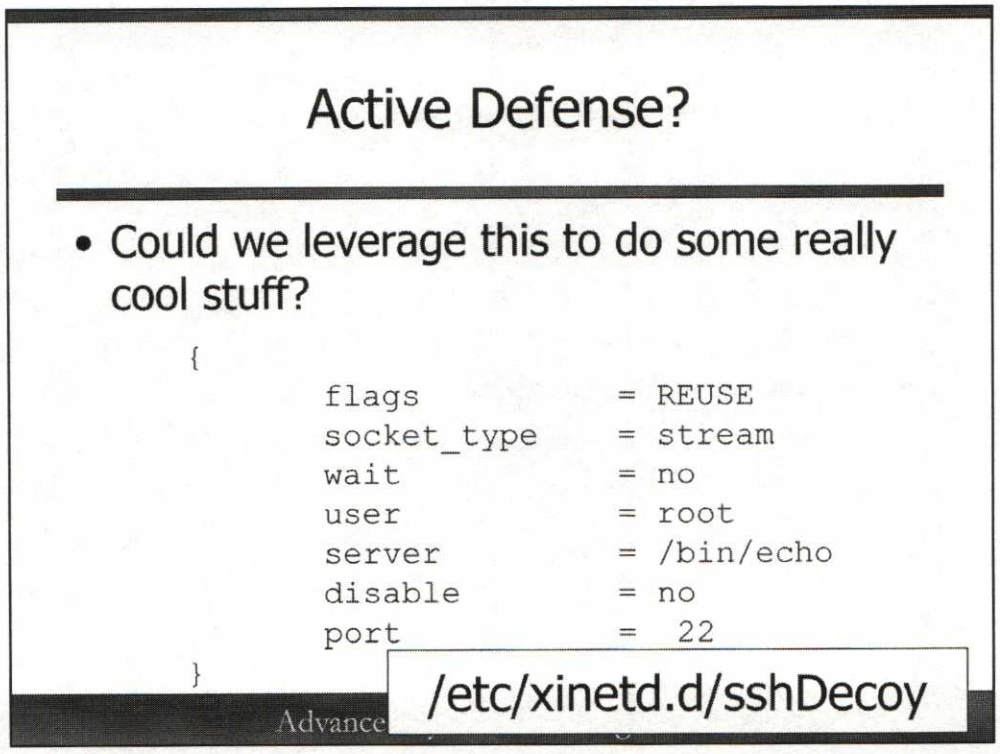


Although most administrators are aware of hosts.allow and hosts.deny and the ability to restrict services based on client hosts, very few are aware that you can take actions when rules match! How is this done?

Notice in the slide that we've configured port 22 to allow connections from any client. Following this, we've added another colon and a shell command. In this case, we're telling the system to send an email to the administrators any time anyone connects to the SSH port.

We could just as easily add rules to monitor common service ports that we are not using. Whenever we see an inbound connection to one of these ports, we can add an iptables rule or a routing command to prevent further activity from this remote system, creating a host-based intrusion detection and even prevention system!

© 2016 David Hoelzer & Enclave Forensics™

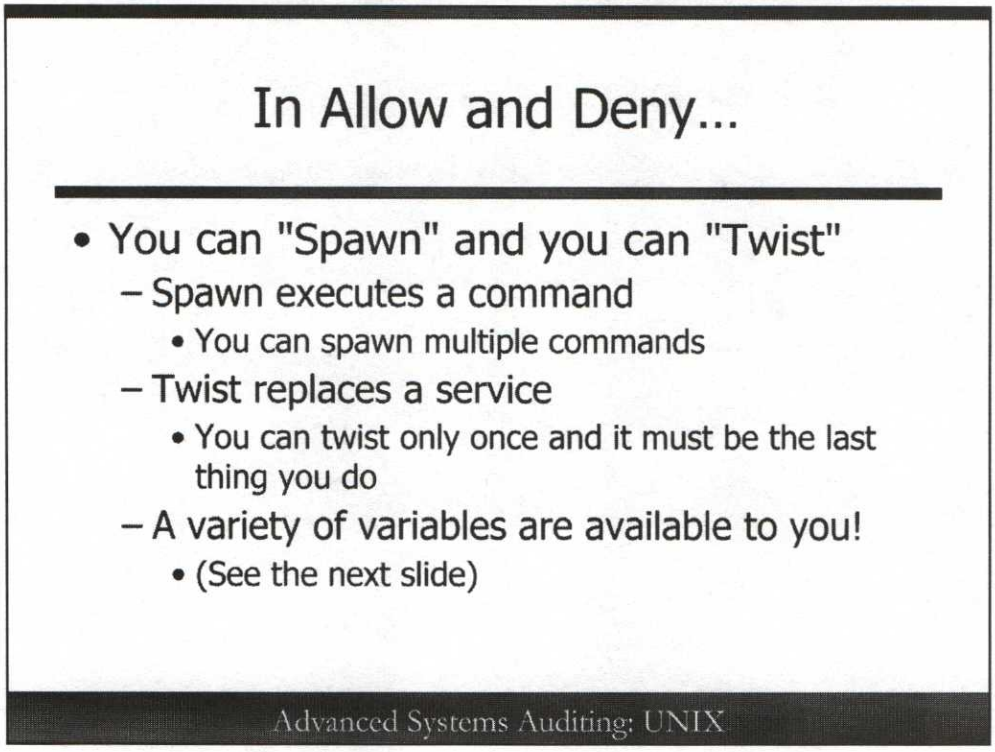


We can actually leverage this feature to do some really cool stuff. For example, in the slide we included a few lines that could be included in your TCPWrappers access controls. When someone connects to our system on ports 21 (FTP), 23 (Telnet), 79 (Finger), or 1080 (SOCKS/Proxies), the system will execute a script named "drop-n-block.sh". What could be in that script?

The script is very simple. It logs the activity so that we know what's happened, and then it adjusts the host-based firewall. The firewall has a rule added to it that will drop all packets originating from that same source host! In other words, if you try to connect to an unauthorized port you are automatically added to a blacklist!!

Just in case you're worried about an automated denial of service, let's think this all the way through. First, if we're applying this only to TCP-based services (remember, TCP wrappers), then the rule will fire only *after* a connection is completed. This means that simply spoofing SYN packets from a partner will not trigger the block. Additionally, if we'd like, we could certainly have some more advanced logic in the script that analyzes who is connecting and makes a more intelligent decision about blocking.

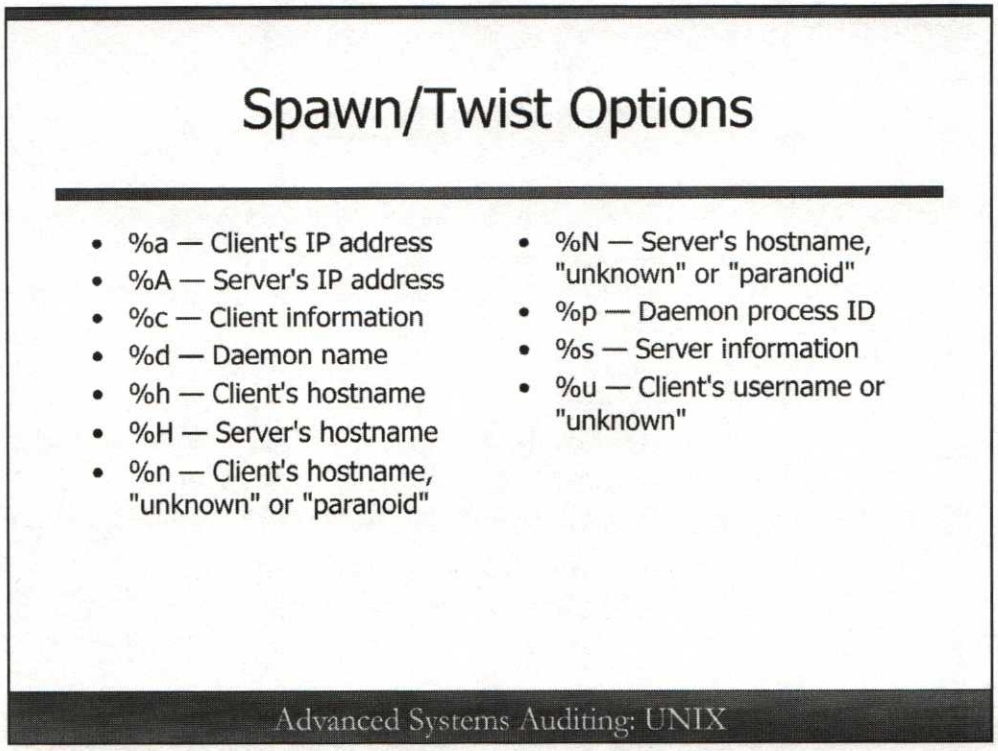
© 2016 David Hoelzer & Enclave Forensics™



Within the allow and deny files, you can actually use multiple 'Spawn' commands for a single connection (which we illustrated previously). The spawn keyword tells TCPWrappers to spawn a subshell executing the script or command that you indicate in the configuration file. Twist, however, is a bit different. Twist replaces the requested service and is more frequently used for creating honeypots (like we are.) Twist, however, can appear only one time on any configuration line, and it must be the very last thing that you do.

Needless to say, if you are choosing to spawn something, you need to be very careful with what that does. In the examples we've given, they block the host and create a log entry, both of which are pretty safe. To assist you with this, there are a variety of useful options available. We have them enumerated on the next slide.

© 2016 David Hoelzer & Enclave Forensics™



These are the variables that are available from a twist or spawn. For the most part, these are self-explanatory. There are a few, however, that deserve a bit more explanation.

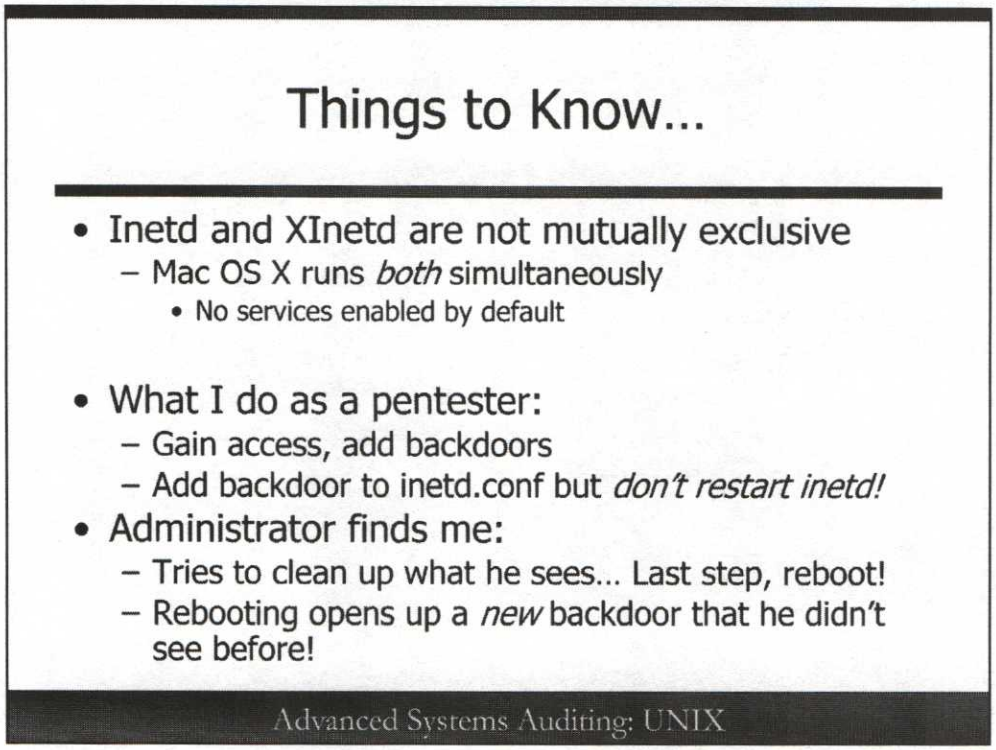
%c will provide, at a minimum, the username and hostname (or IP address, if unavailable). Some platforms might provide more data.

%s will return, at a minimum, the daemon process number and the hostname (or IP address, if unavailable) of the server.

%h and %H will provide the hostname only if it is available. If it cannot be found, "unknown" will be returned.

Using these arguments, you can automate just about any reaction that would be meaningful. These certainly provide useful mechanisms both for active defense and for continuous monitoring.

© 2016 David Hoelzer & Enclave Forensics™



There are some interesting things to know about wrappers, inetd and xinetd. One of the most curious is that inetd and xinetd are NOT mutually exclusive. This means that both tools can be used to mediate access on the same system! A perfect example is Mac OS X. Both inetd and xinetd are started at boot time, though inetd usually dies because there are no services turned on by default. What this means for you and I as security auditors is that it is not sufficient to see that there's an xinetd.conf file and assume that xinetd is being used to control all services! We must verify that however a service is being offered, there are adequate controls in place around the service.

The reason that we point this out to you has to do with the process that I use as a pentester. When we pentest an organization, we're looking to see whether or not we can get in and to measure how long it takes the security organization to detect us and respond. For this reason, when I gain access to a system I'll create my backdoors on UNIX systems and will, additionally, create an entry in the inetd.conf file to open another backdoor. I will not, however, restart inetd! Instead, if an administrator finds me later, he might try to remove what he can see. Most likely, he will do this by trying to identify ports that are listening that he doesn't expect to find. Great!

The problem is that his very last step will often be to reboot the system… Can you see the trouble? He didn't "see" my backdoor before because it wasn't open. As soon as he reboots, however, he immediately opens up a *new* back door!!!

© 2016 David Hoelzer & Enclave Forensics™

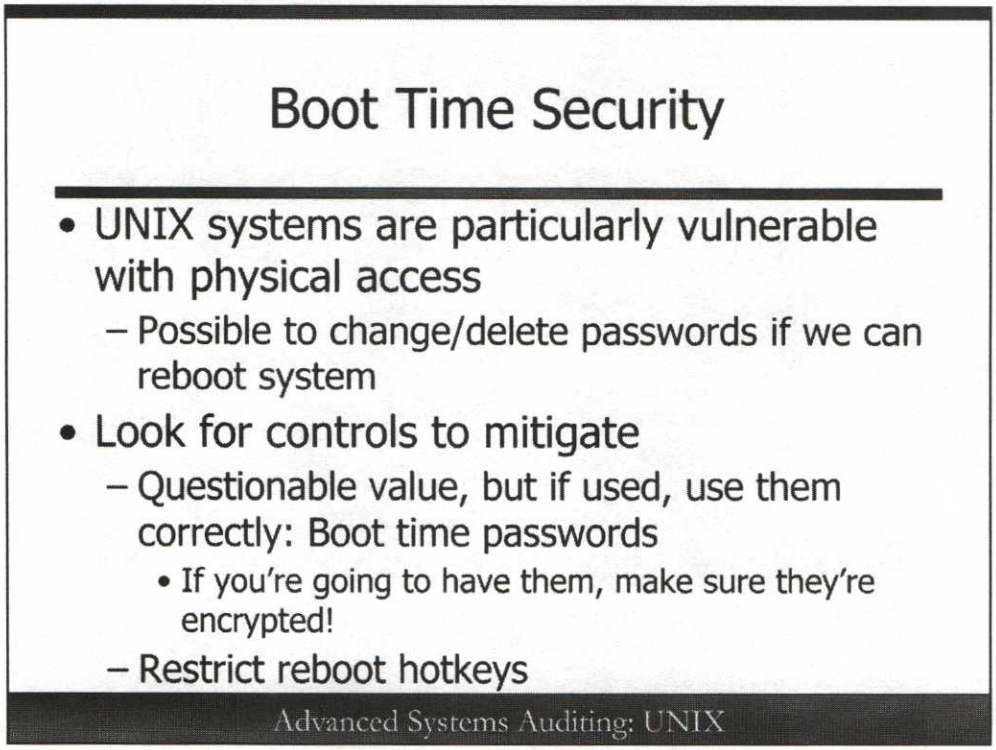
Audit Objective

- Objective: User Management/Access
  - Ensure unique user accounts
  - Identify authorized users
  - Examine password settings
  - Ensure strong passwords are used
- Audit Activities:
  - Examining /etc/passwd, /etc/shadow
  - John the Ripper

Advanced Systems Auditing: UNIX

Closely connected with authorized/unauthorized access issues are the user management and access control issues. We need to ensure that all of our users have uniquely identified accounts and that access is restricted to authorized users only. We need to make sure that only strong passwords are used and that the password control settings match our policies. To do this, we will examine several different tools and files.

© 2016 David Hoelzer & Enclave Forensics™

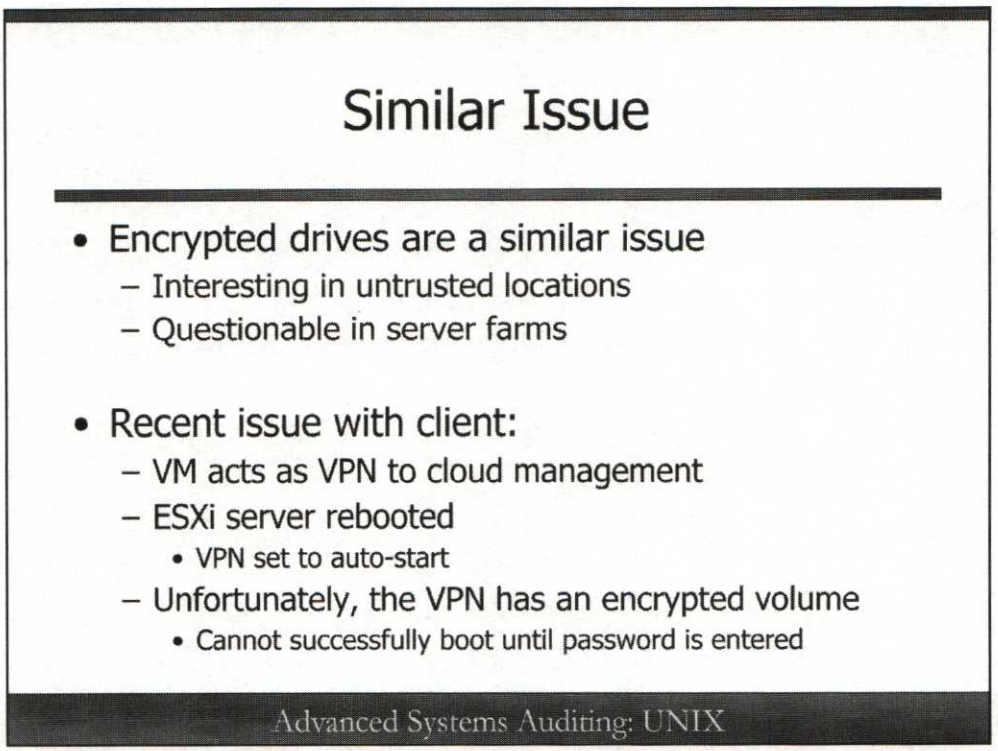


Most systems are vulnerable should an attacker obtain physical access to the system. Aside from being able to outright damage or steal the system, UNIX systems are particularly vulnerable in that deleting or changing passwords is particularly trivial if it is simple enough to reboot the system. For this reason, we would like to examine whether or not there are local controls to mitigate this.

The main types of controls that we would look for are boot time passwords to prevent easy access during a reboot and restrictions for any hot keys that could bring up a system monitor or reboot the system automatically.

These are not cure-alls, of course. It is still simple enough to remove the drive and mount it on another system or to perhaps boot off a CD; however, they are surely low stone walls.

© 2016 David Hoelzer & Enclave Forensics™

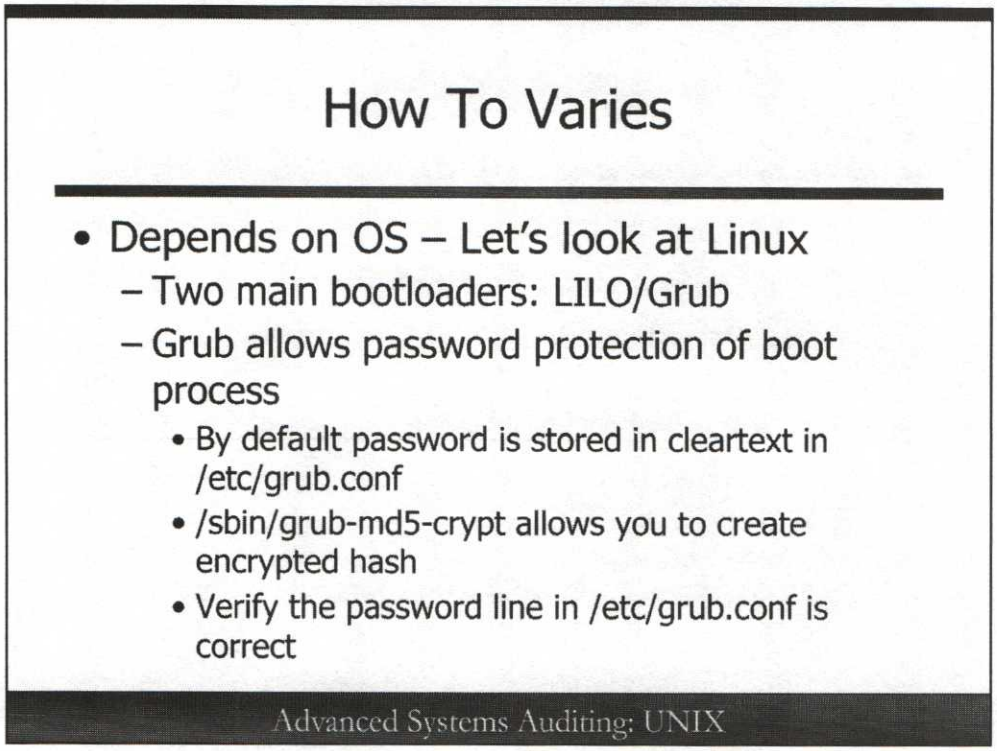


A similar issue can be created when an administrator decides to build a server with an encrypted volume. Although encrypted volumes can be quite powerful when the system is operating in an untrusted environment where physical theft or intrusion is likely, it makes far less sense when we are talking about a server in a locked cage in a secured datacenter. What's the risk? We can illustrate it with something that recently happened with one of my clients.

The client leases several ESXi servers in a secure datacenter in another part of the country. To improve the security of the deployment, each ESXi system has a VPN/FW VM running on it. All of the management functions for ESXi have been moved onto a private network behind the FW/VPN, requiring that administrators first connect to the VPN before administering the ESXi system. To facilitate this, in event of an outage, the VPN/FW VM is set to automatically start 30 seconds after the ESXi system finishes booting up.

Unfortunately, the administrator who originally set up the VM failed to document that he had built it using an encrypted volume. This means that the correct password must be entered before the VM will boot successfully. Of course, this was discovered when there was an actual outage, leading to hours of work. First, remote console access to the ESXi server had to be obtained. Next, the undocumented password had to be located. Finally, the VM could be brought back online.

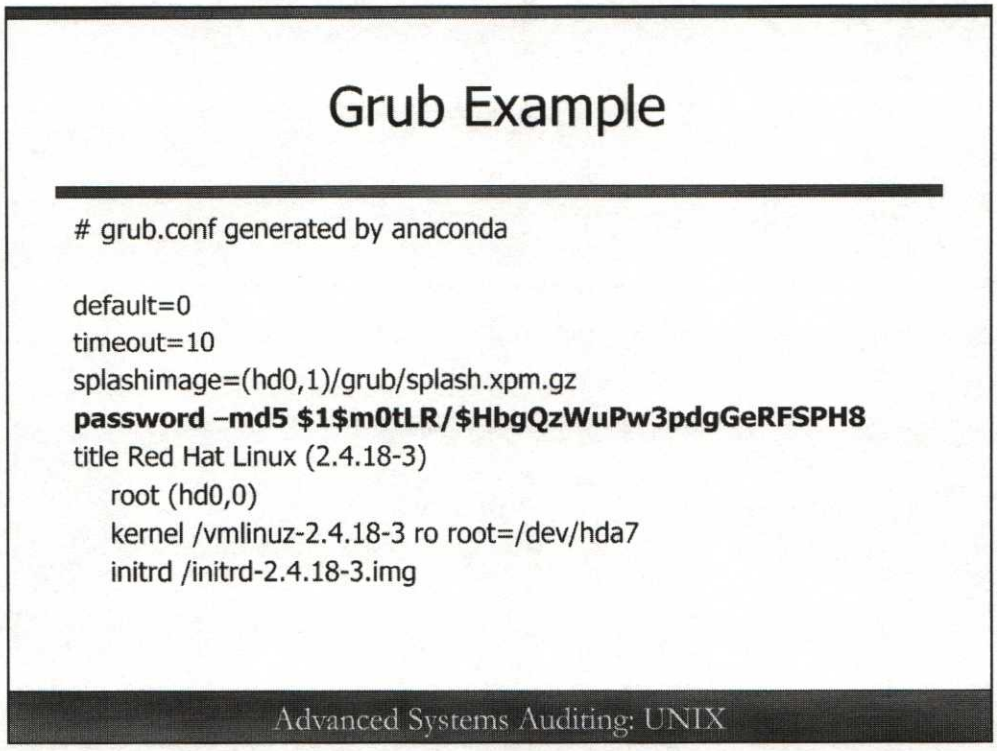
© 2016 David Hoelzer & Enclave Forensics™



How to verify that there is a password required during boot time varies by system. Let's use Linux as an example.

If we have a Linux system, particularly one with the Grub boot loader, then during installation the administrator is given the opportunity to create a password that must be entered to boot the system. We could verify that this is the case by rebooting the system, but we can also verify by examining the grub configuration file. This is actually more effective because the default is for the password, if it exists, to be stored in clear text in the grub configuration file. This means that the best verification is to examine the grub configuration file directly. Let's look at an example on the next slide.

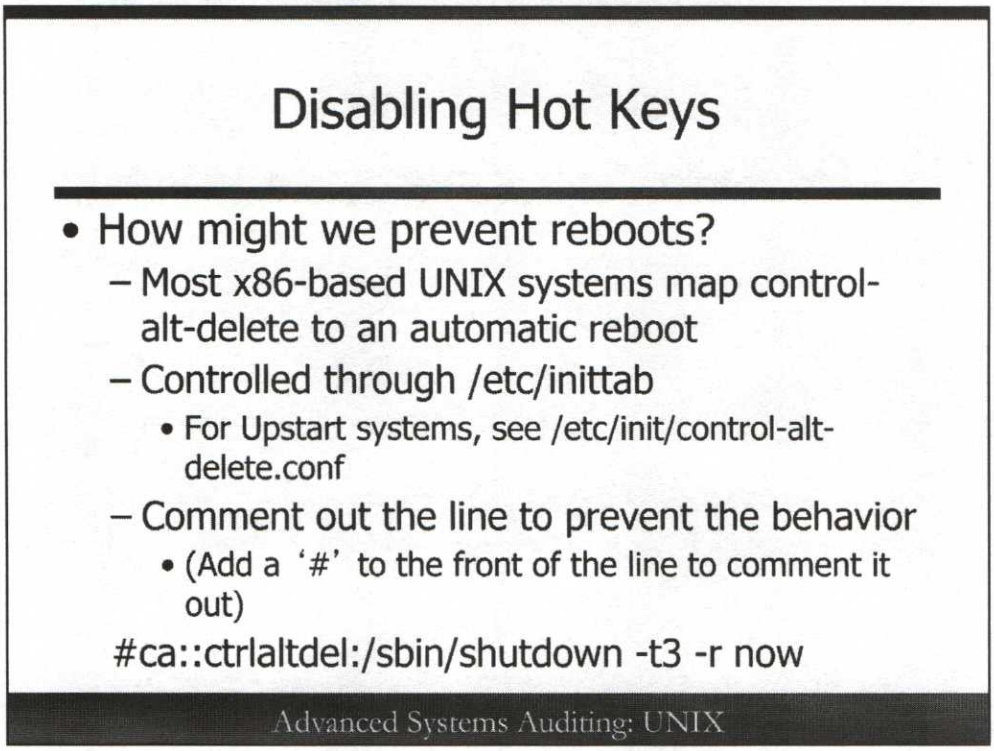
© 2016 David Hoelzer & Enclave Forensics™



In this slide, we can see an example Grub configuration file. Included in the configuration are options to decide which boot image is the default, how long the boot screen should be displayed before proceeding, what splashpage to display while waiting at the boot screen, and so on. We can also see the first bootable system description, which begins with the "title" line. The lines beneath that define everything about the Red Hat Linux system, which is the default boot image. Notice the line in bold, however.

If a password is configured, this line will exist. If the password is encrypted, then it will also include the –md5 option. Could we identify whether or not that line exists and the password is encrypted through a script?

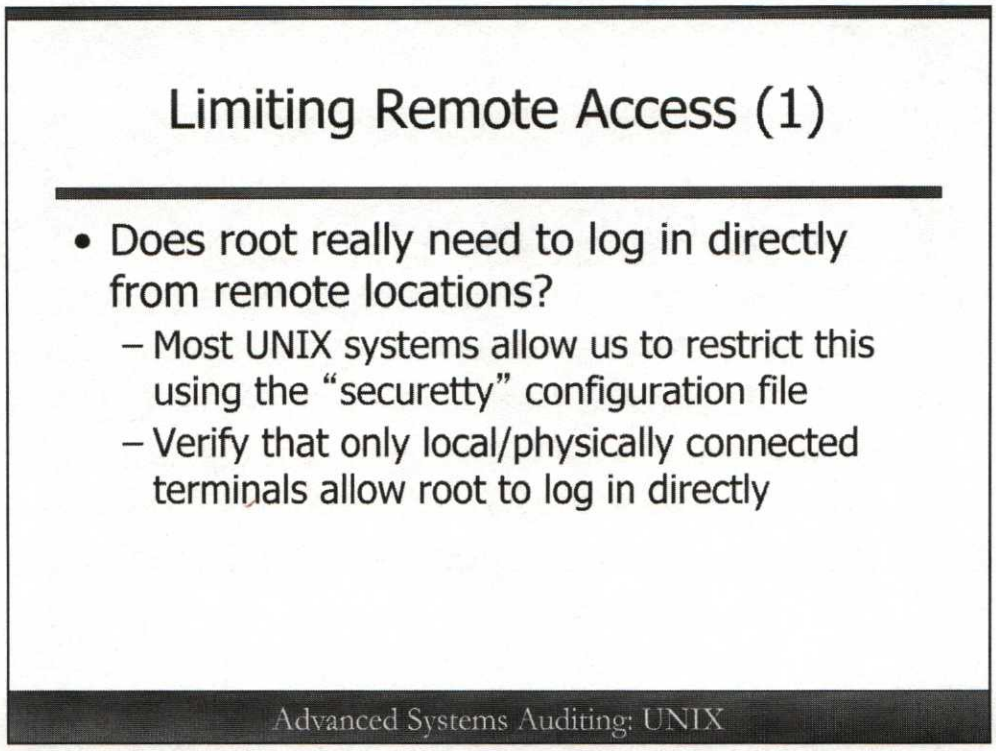
© 2016 David Hoelzer & Enclave Forensics™



We also indicated that it would be good to disable hot keys that perform automatic reboots. This won't prevent someone from pulling the plug, but hopefully our datacenter is just a bit more secure than average and servers are kept within locked cages and consoles are locked with passwords. Even in this scenario, someone could press control-alt-delete to initiate a reboot on most x86-based UNIX systems, but this can be controlled as well.

The /etc/inittab file usually will contain a line that begins with the letters "ca" that is used to define what actions to take when these keys are hit. If we remove or comment out this line, it will no longer be possible to initiate a reboot simply by pressing control-alt-delete. Instead, it will now be necessary for root to log in and manually run the reboot command.

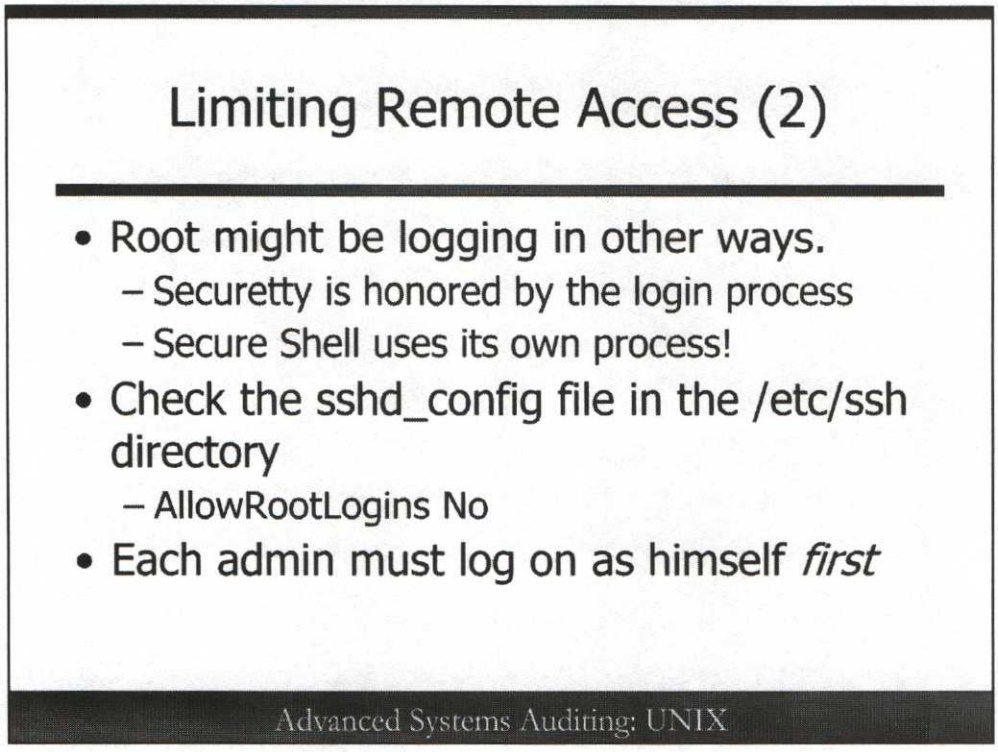
© 2016 David Hoelzer & Enclave Forensics™



Another item that falls into this section is controls that limit the access to certain accounts from certain places. For instance, should users be able to log into the root account directly across the network or would you prefer that they first authenticate as some other user? To allow for some control here, most UNIX systems allow this to be controlled through the use of the /etc/securetty file. This file contains a list of all of the ttys (teletypes… yes, we know that we don't use teletypes anymore, but UNIX has been around for a long time. Think of these as endpoints) that are considered secure enough to permit the root user to log into them.

For our audit testing, we should verify that the only terminals that permit direct logins by the root user are physically connected to the server. These days that usually means that they are at the console itself.

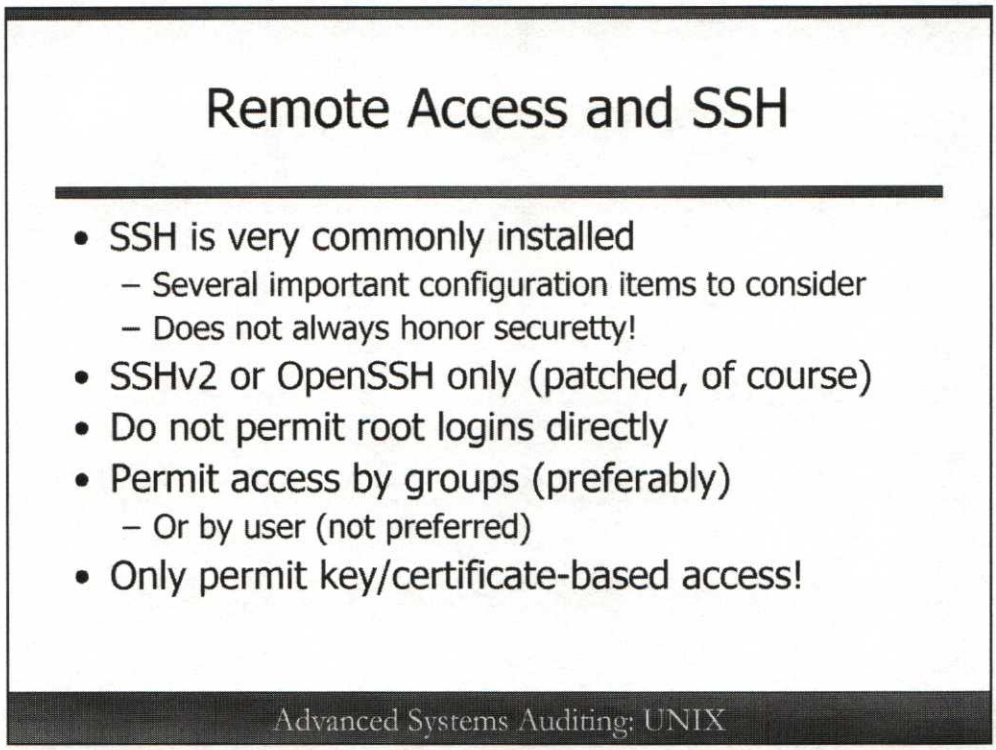
© 2016 David Hoelzer & Enclave Forensics™



Although the securetty file is important, it's also somewhat deprecated. Hopefully, it will be rare for you to find telnet running on a UNIX system. (That's my hope, anyway…) The replacement for telnet is Secure Shell. This is a dramatic improvement because we're now strongly encrypting the connection. However, Secure Shell sidesteps some of our typical login and authentication processes.

This isn't really a bad thing, but it's certainly something to be aware of. To prevent root from logging in directly over SSH, you should find an AllowRootLogins line in the sshd_config file that is set to "No."

© 2016 David Hoelzer & Enclave Forensics™



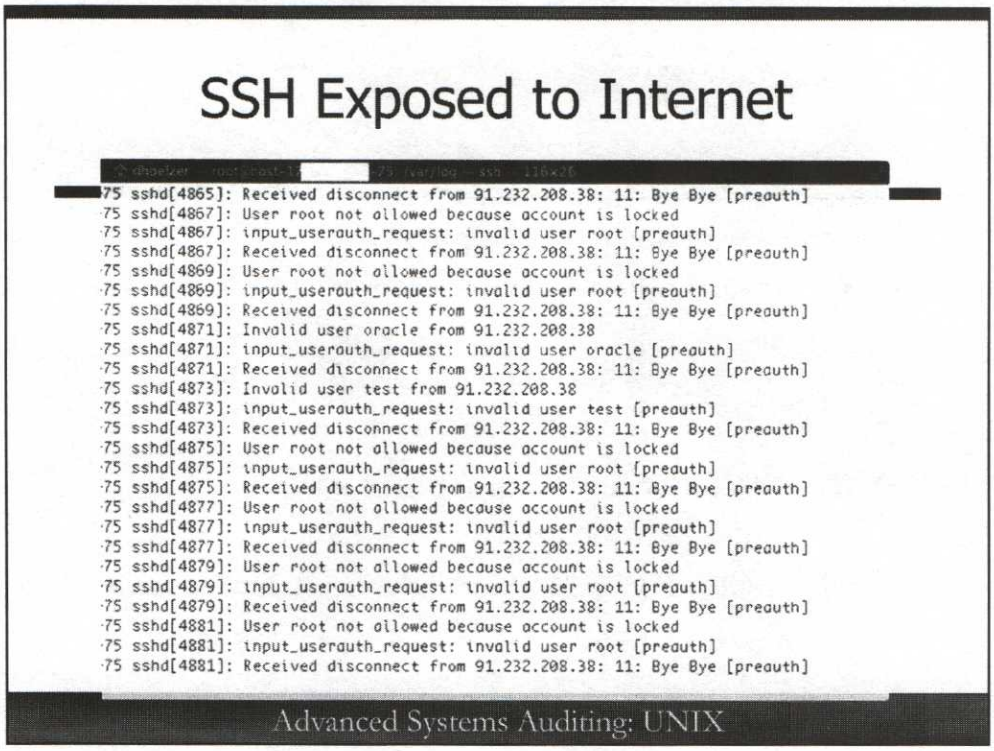
Let's add a bit more detail for SSH. SSH, or Secure Shell, is often spoken of as the secure replacement for Telnet. It is certainly that, but it is far more. SSH is also used for tunneling, proxying, secure file access, and more.

If at all possible, any remote administrative access that requires a command line should be performed over an SSH connection. Not only does it provide strong encryption, but it also can provide for strong authentication. There are still some items to look for during an audit, however.

First, we should find that the system is running a current version of OpenSSH or SSH and that only SSHv2 or the OpenSSH protocols are supported. If any older version is used, the protocol is susceptible to a man-in-the-middle attack. Another important item is preventing a user from logging in as root directly. We must require that users log in with a named account first, allowing us to have a level of accountability.

Next, access to the system should be restricted. Simply installing and turning on SSH will allow any valid user to authenticate. Instead, only specific users or, preferably, groups should be permitted to use SSH. For those individuals who are permitted access, you would like to find that only public keys or certificates are used for authentication, never passwords. Why not? Take a look at the next slide…

© 2016 David Hoelzer & Enclave Forensics™

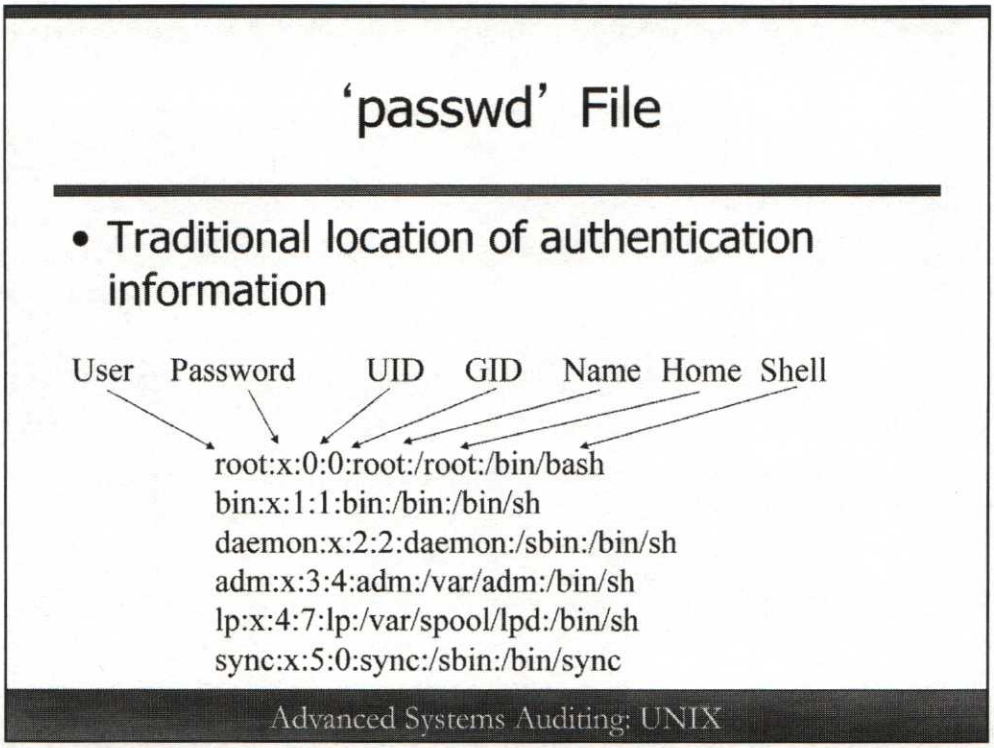


If you have a system running SSH connected to the Internet, you should have a look at the logs. Unless you are restricting access to the SSH service based on a firewall rule, you will find tens to hundreds of thousands of login attempts to common accounts.

When you first see this, you might wonder why attackers would do this. Do they really expect to find accounts exposed with weak passwords? To answer that question, consider this: Why would a large number of attackers perform this same type of scan? The only reasonable answer is that they are meeting with success! Systems with default accounts and default passwords are rampant. Systems with weak passwords are even more common.

Requiring key- or certificate-based authentication eliminates this exposure.

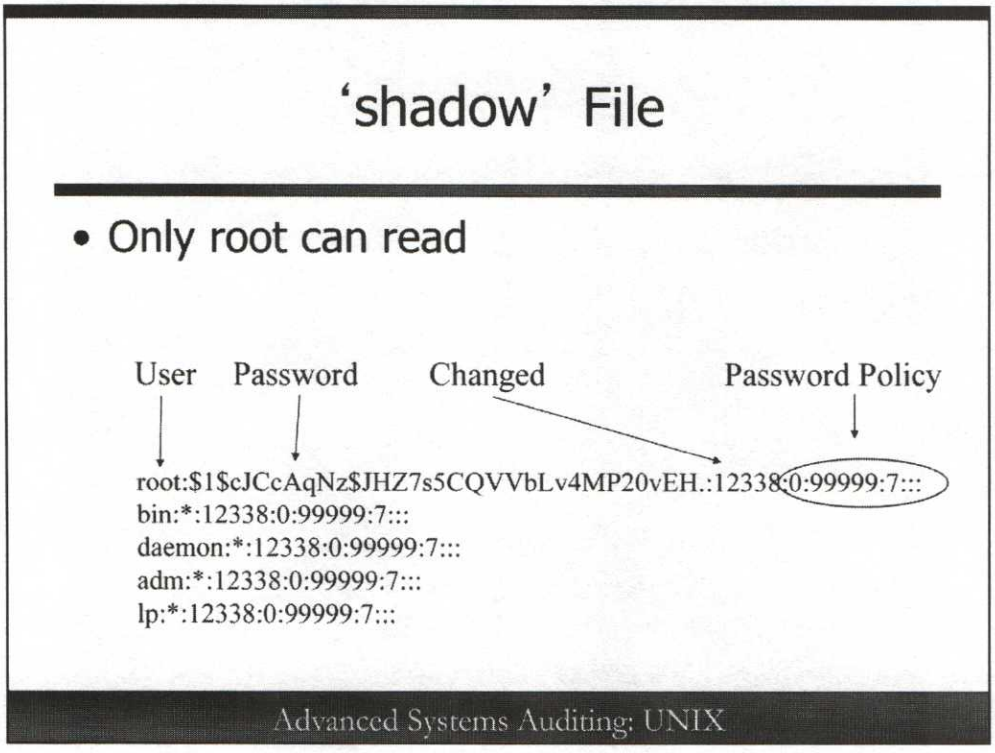
© 2016 David Hoelzer & Enclave Forensics™



The password file is the traditional location of user credential information. The file is, by default, world readable. This is done to permit the system to correlate user IDs to usernames when looking at directory listings and process lists.

In the slide, we described each of the various fields in a typical password file. We begin with the username in the first column and the password in the second column. Notice that in this case all of the passwords are marked "x". This column is actually where the encrypted password would reside. If this field is blank, then there is no password. Seeing all of these fields marked with an "x", however, indicates that the encrypted password hashes are stored somewhere else, which we will see on the next slide. The next column is the actual user ID number followed by the primary group ID. Next, we have the "real name" or more descriptive name for the user, and then the home directory into which the user is placed when he logs in, and finally the default shell that is run when the user logs in.

© 2016 David Hoelzer & Enclave Forensics™

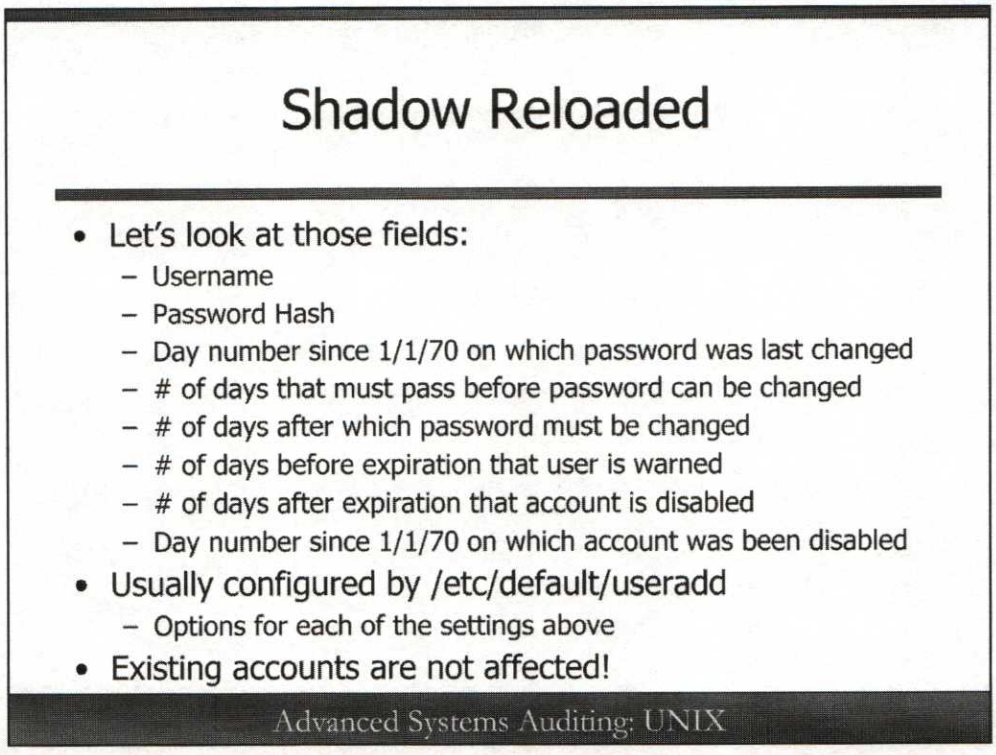


The shadow password system was created first to eliminate the readability of encrypted password hashes but secondly to provide alternative means of encrypting the passwords. In this case, for instance, the encrypted password hash is an MD5 style password hash rather than the typical DES hash.

Notice that many of the password fields have an '*' in them. Just as in the passwd file, if the field is blank, it indicates no password, but any other character represents a password. What is the significance of a single character? Well, because the password hash function will always create the same number of characters (definitely > 1), this represents a password that can never be matched. In other words, the account is not disabled, but it is impossible to log into the account directly!

Other additions to the typical passwd file are the inclusion of the post epoch day on which the password was last changed followed by the password policy information, which includes the minimum password age, maximum password age, expiration warning timer, post expiration disable timer, and a count for how many days an account has been disabled. Quite honestly, very few people, including administrators, actually know how to interpret a shadow file, so you're already a leg up on them!

© 2016 David Hoelzer & Enclave Forensics™

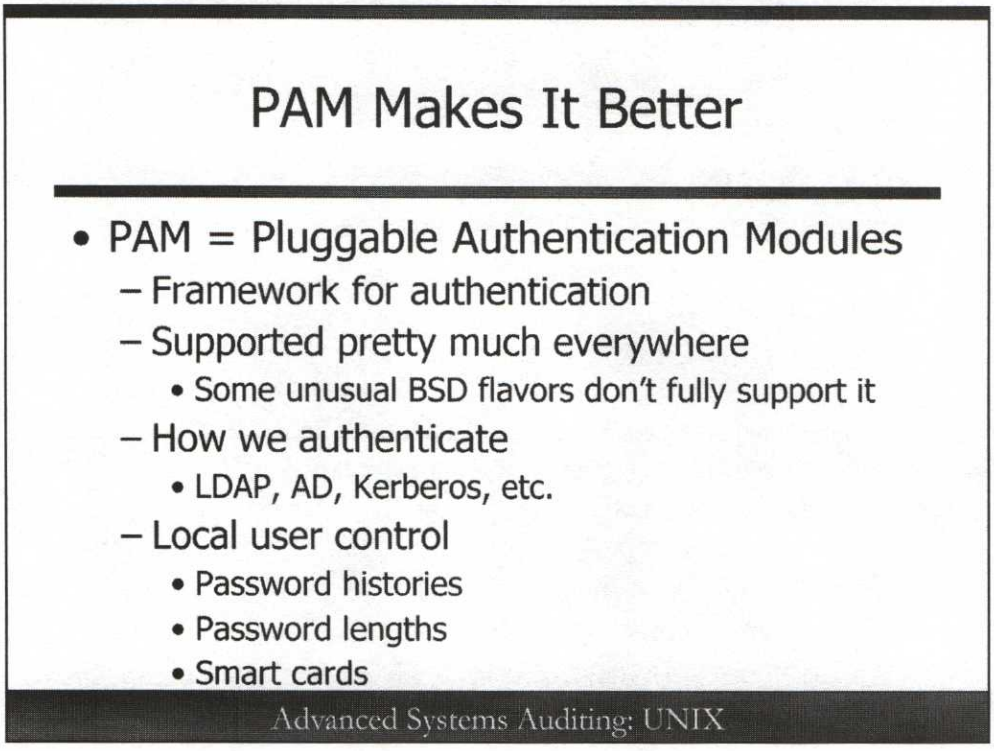


As you can see from the slide, there are a number of fields in the Shadow file that can be used to configure password controls for user accounts. In our experience, however, UNIX systems are typically configured to allow zero days between changes and 99,999 days between changes, which effectively turns off password policies.

It is also useful to know that even after the account has been disabled, it is possible for cron jobs and at jobs to continue to run with this accounts credentials. In fact, it is usually still possible to use the account to FTP into the system remotely and even to use the "su" tool to change to this user ID. Disabling the account effectively prevents only console-based access to the system.

If your organization requires password change controls, you will want to look not only at the shadow file itself for evidence that the settings are correct but also at the configuration file for the useradd tool (or whichever tool is used in your environment for the creation and management of user IDs on your UNIX system). Typically, this is found in the /etc/default directory. Please note that if an administrator changes the default settings, it has no impact on the existing accounts! They must be modified manually.

© 2016 David Hoelzer & Enclave Forensics™

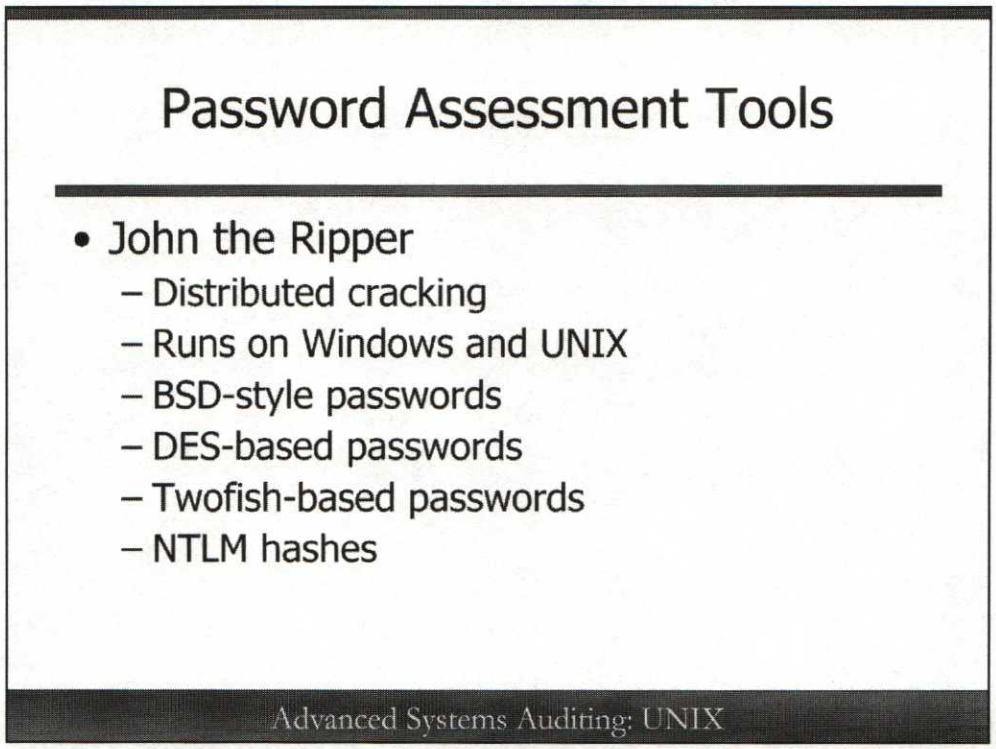


As an alternative to the traditional passwd/shadow files on our UNIX systems, we can leverage a product called "Pluggable Authentication Modules (PAM)." This is a framework that provides various authentication modules and controls that can be applied to our systems. Among other things, PAM can be used to configure our UNIX systems to use Active Directory credentials for users logging onto the systems for administrative or other purposes. This is a very powerful feature because it not only allows us to reuse a trusted credential store (good security principle), but it also means that the password policies that we are enforcing in our Active Directory are now also being applied to our UNIX systems.

Consider the side effects. For example, password resets are now being *enforced*. Password complexity is now consistent across the organization. Password histories, which are otherwise unavailable in UNIX, can now be enforced. Perhaps the best feature, though, is that by connecting our authentication to our Active Directory, we need to disable a user in *only one place!*

To be clear, we're not saying that you should go back and insist that your enterprise re-architect all of the systems that you have. We are suggesting, though, that you might have the ability to add some very cool centralization and management controls at an extremely low cost.

© 2016 David Hoelzer & Enclave Forensics™



Of course, no audit would be complete without running a password assessment. One of the SANS Top Ten Vulnerabilities was User Accounts With No or Weak Passwords. We recommend using a password assessment tool randomly, and in conjunction with password filters that permit only "strong" passwords. The threat of cracking is usually enough to make people choose good passwords in the first place.

Yesterday, we looked at a Windows-based password-breaking tool that will also handle UNIX passwords. Today, we give you a UNIX-based tool that not only can break Windows passwords, but can also run on Windows computers! In fact, this tool also allows you to perform distributed password-cracking attacks, speeding up the entire process. John is also capable, through add-on modules, of breaking just about any kind of password hash that you'll come across making it a pretty versatile tool.

© 2016 David Hoelzer & Enclave Forensics™

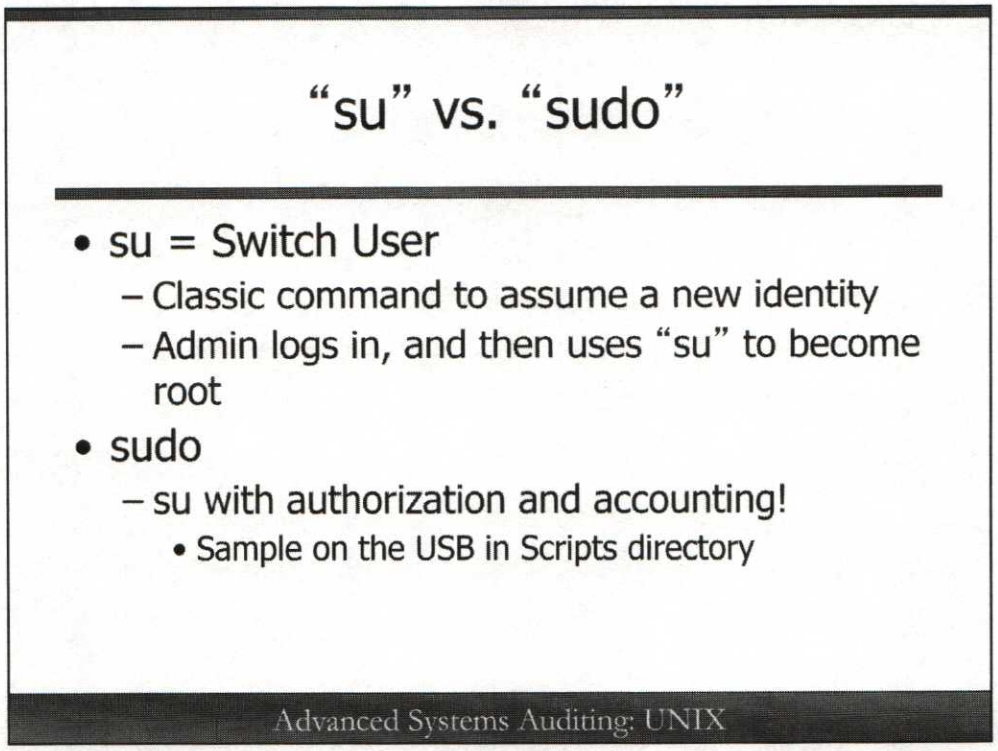
Audit Objective

- Objective: Unauthorized Access
  - PoLP for users
  - Ensure only necessary files have set-user or set-group bits set
  - Identify recently modified files
- Audit Activities:
  - Find
  - Ls

Advanced Systems Auditing: UNIX

Next up in our audit objectives is examining files, file settings, and file activities. To do this, we can use some handy UNIX tools like 'find' and 'ls.' Although we've already seen 'ls' earlier in the day, there's a great deal of capability in the 'ls' tool and we'll have a look at a bit more of it here.

© 2016 David Hoelzer & Enclave Forensics™

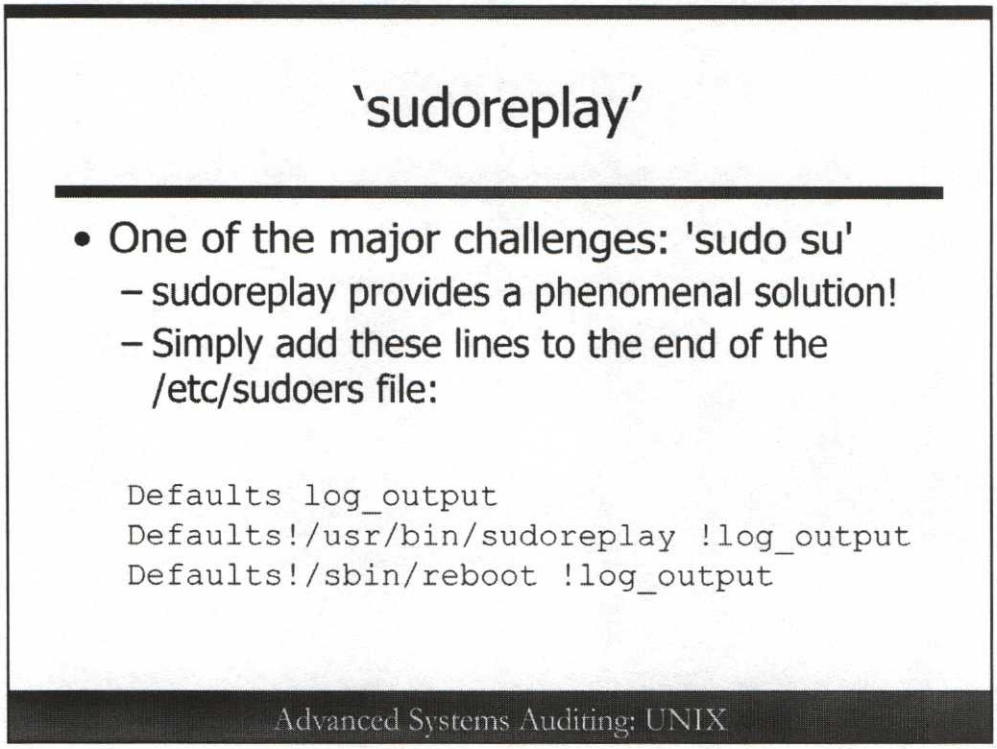


During the course of this class, it has already been mentioned that administrators should never be permitted to log in directly as "root" or "administrator." Instead, they should be required to log in using a personal account, and then become the administrator. The typical way that this has been done on UNIX systems over the years is using the "su" tool to Switch User.

A much better way to accomplish this is using a standard replacement for su: sudo. You can think of this command as meaning "su Do", which is what's actually happening. With sudo, the administrator logs into the system using his personal account. When he wants to perform an administrative activity, he must enter "sudo <command>". For example, to kill a process as root, he could run, "sudo kill -9 12345".

The beautiful thing about this system is that the user can optionally be prompted for a password before the action is taken. The password that the user must enter is *his own* password, not the root password! Even better, every action taken using sudo is logged, creating useful accounting records. Finally, the system can be configured to create groups of users with different authorization requirements and permission to run specific tools in specific ways. You can even restrict which commands can be run. For example, you can allow a user to run "sudo su another_user" while preventing him from running "sudo su" or "sudo su root" or any variation thereof!

This tool is a must-use for administrators. Without something like this, there is no real way to account for activities on the system.

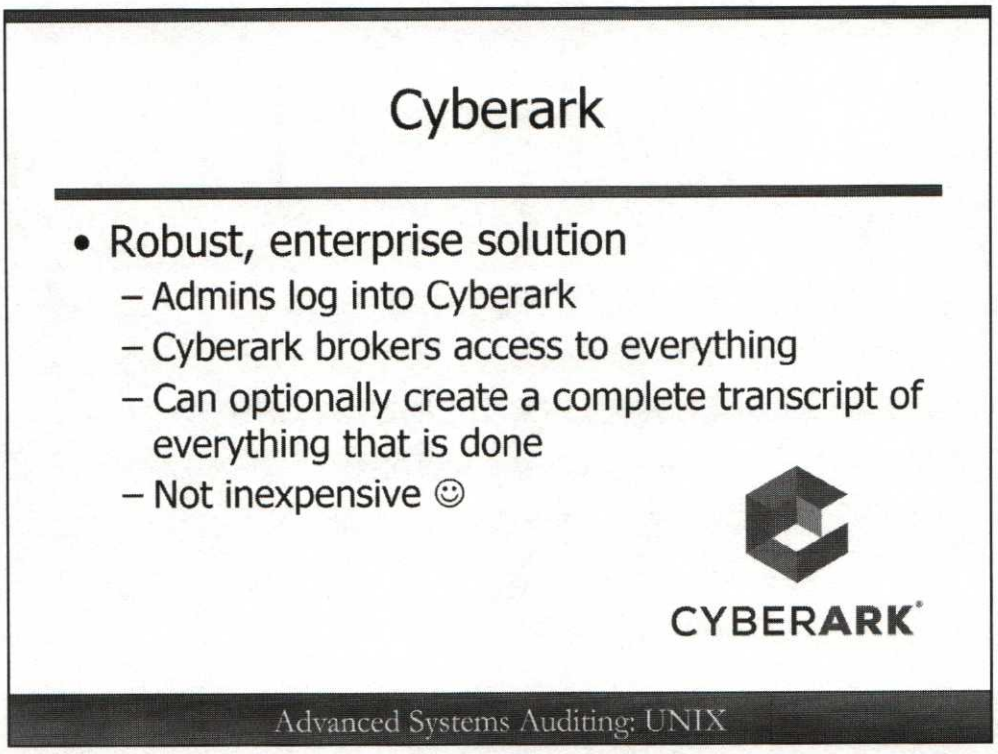


'sudoreplay' is an extremely useful tool that, if configured, can be super for auditing administrative activities. Even with 'sudo' being used to track and audit administrator activities, it's a pretty simple matter for the administrator to either 'sudo su', becoming root, or escalate to root through some other means. With the addition of three simple lines to the /etc/sudoers file, however, the system will generate automatic audit trails for everything done under an 'sudo' spawned shell!

To view the replay, you can first execute 'sudoreplay –l' to obtain a list of all of the sessions that are available. You can also provide a search expression, allowing you to search through all of the transcripts of sessions for a particular command or user, for example. In the listing, you will find an "ID" that can then be used to replay that session.

When you now execute 'sudoreplay <ID>', the session will begin to replay in real time! You can also press the space bar to pause, '<' to halve the playback speed, and '>' to double playback speed! We have an example in the lab exercises. It's truly an amazing tool!

© 2016 David Hoelzer & Enclave Forensics™

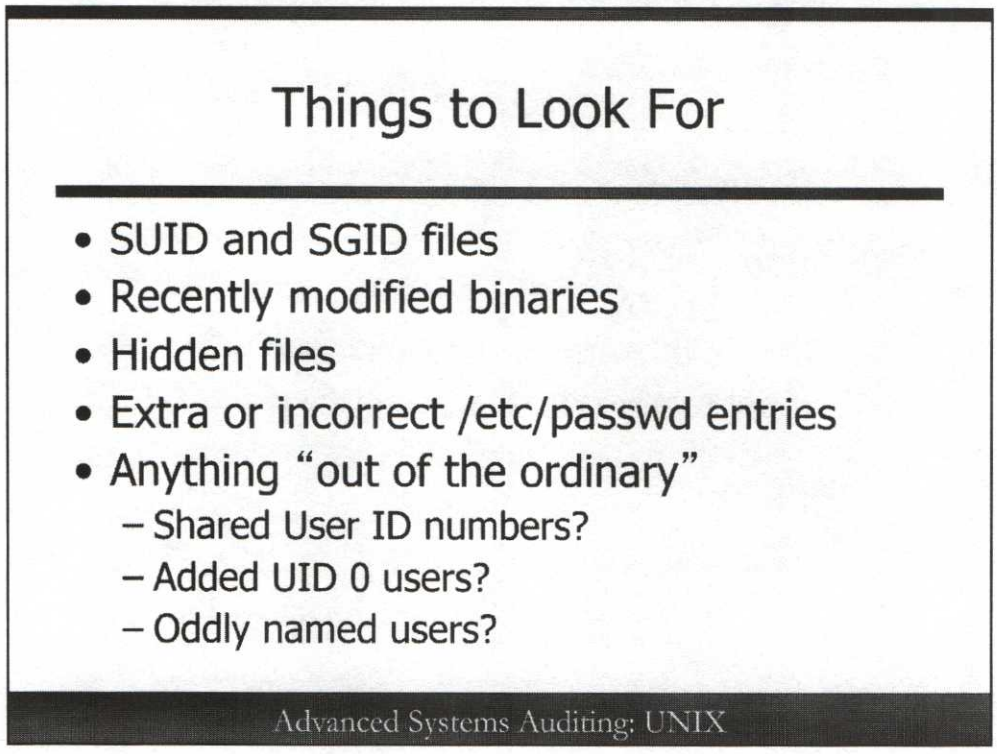


The problem of auditing administrator access to systems is not a new one. The sudo tool provides a very realistic and inexpensive mechanism that can be implemented to begin accounting for these activities. The addition of sudoreplay makes it even more powerful. Even so, this is a solution only for our UNIX systems.

The idea of leveraging something like PAM to rely upon centralized authentication into an LDAP or perhaps our AD infrastructure also makes a great deal of sense from an enterprise perspective. This reduces the effort required to simply verifying that the proper groups are being enforced in the directory and that the UNIX systems have been properly configured to enforce authentication requirements to the directory.

A larger scale more holistic solution does exist, however. Cyberark is an example of just such a tool. Although the price tag can be high, this provides a single tool that can be used to enforce administrative access controls across both your UNIX and Windows systems. It also provides an even better implementation of the sudoreplay concept, allowing you to record and later play back any and all administrative actions within your systems.

Using a system like Cyberark, none of the administrators actually know any of the administrative passwords. Instead, their connections are brokered through Cyberark, which then creates the authenticated session using the actual credentials of the protected system.



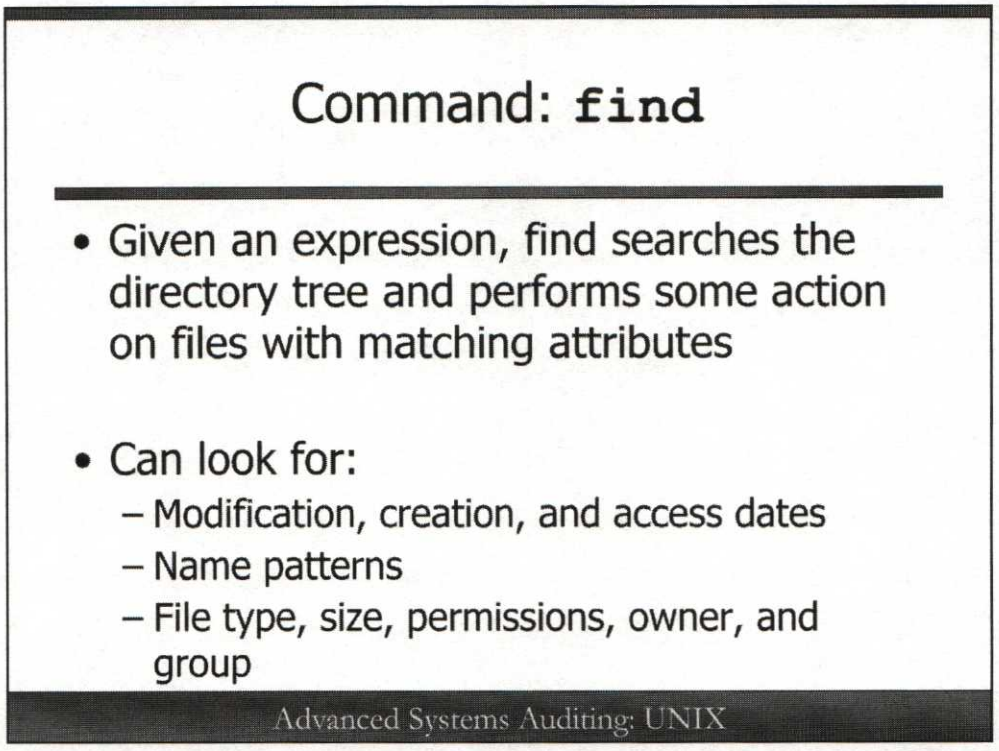
OK, so what else can we look for that might be indicative of compromise? SUID and SGID files are often exploited so we'll take a look at those. We should also be concerned with any binaries that have changed recently or hidden files on the system. We would also, generally, be interested in anything "odd."

It's unfortunate that we can't definitively define everything that would be "odd," but we can certainly give some examples. One of these would be shared user accounts.

In UNIX systems, users really are not identified by names. The usernames are purely for our convenience. The system itself is concerned only with the UID number (the second field in the passwd file). If we find two user names that have the same UID, the administrators have effectively created two *passwords* that are equivalent for the UID. This is actually a common artifact of a compromise, too. It is not unusual to find that there is an additional username assigned to UID 0, effectively creating a second password for the root user.

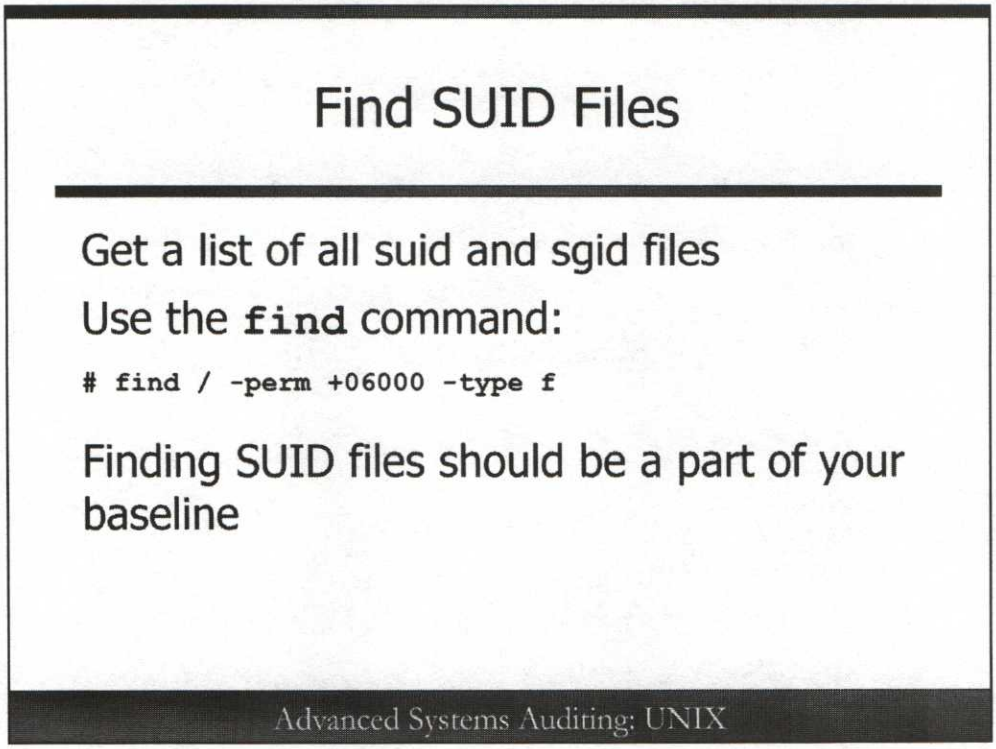
Another "odd" thing to look for are strangely named users. Most organizations use some sort of standard when it comes to naming accounts. Typically, it is not difficult to just "see" when an account name is odd. For example, an account named "dave" rather than "dhoelzer."

© 2016 David Hoelzer & Enclave Forensics™



Find is an incredible command. The degree to which one can define filenames and file classes is astounding. And better yet, for each matching file that is found, some action can be performed. This is as incredibly useful as a system maintenance feature, but is also incredibly dangerous. Be careful performing some action on files returned. We've messed more than one system "rm-ing" the results of find.

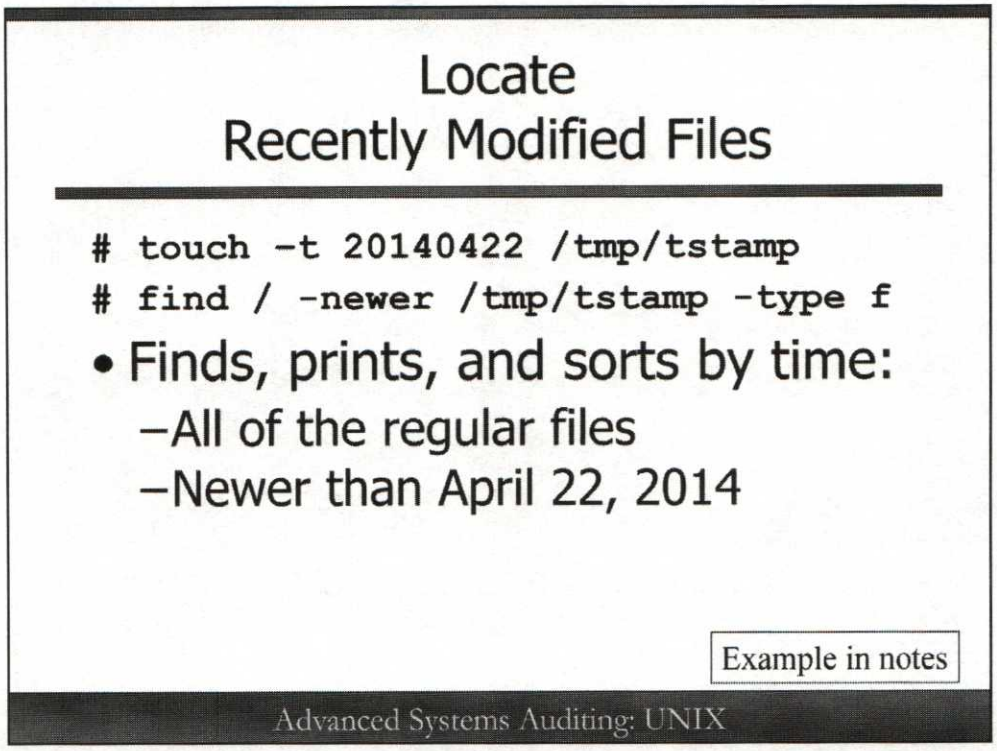
© 2016 David Hoelzer & Enclave Forensics™



To audit suid root files, use the find command with a few extra parameters. This incantation tells find to start at the root directory, /, and recursively search the file system looking for any files that are suid or sgid and of type file.

One of the first things that I do when I suspect that a system has been compromised is to check the current list of suid files against the master that I created and stored on a CD, USB stick, or read-only network share. Why do I store the suid file listing on some other device or system? This is a special measure that should be taken to ensure that the contents of this file cannot be altered by a remote hacker.

© 2016 David Hoelzer & Enclave Forensics™

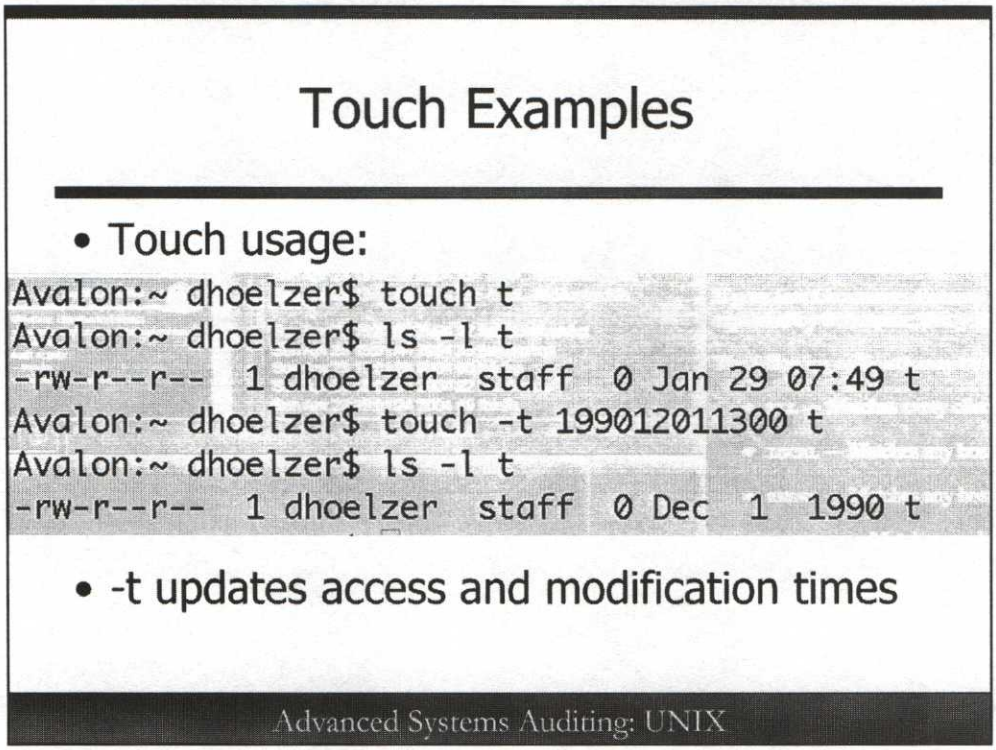


Another fantastic way to use Find is to locate recently modified files. Find allows you to search for files by comparing timestamps, looking for files that are newer or older than another file. In the example in the slide, we use the 'touch' tool to create the file to compare to.

The touch command exists primarily to update timestamps, usually to the current date and time. It originally allowed developers to update timestamps on files to force them to be recompiled. This was especially important years ago when we did not have really great time synchronization tools like NTP.

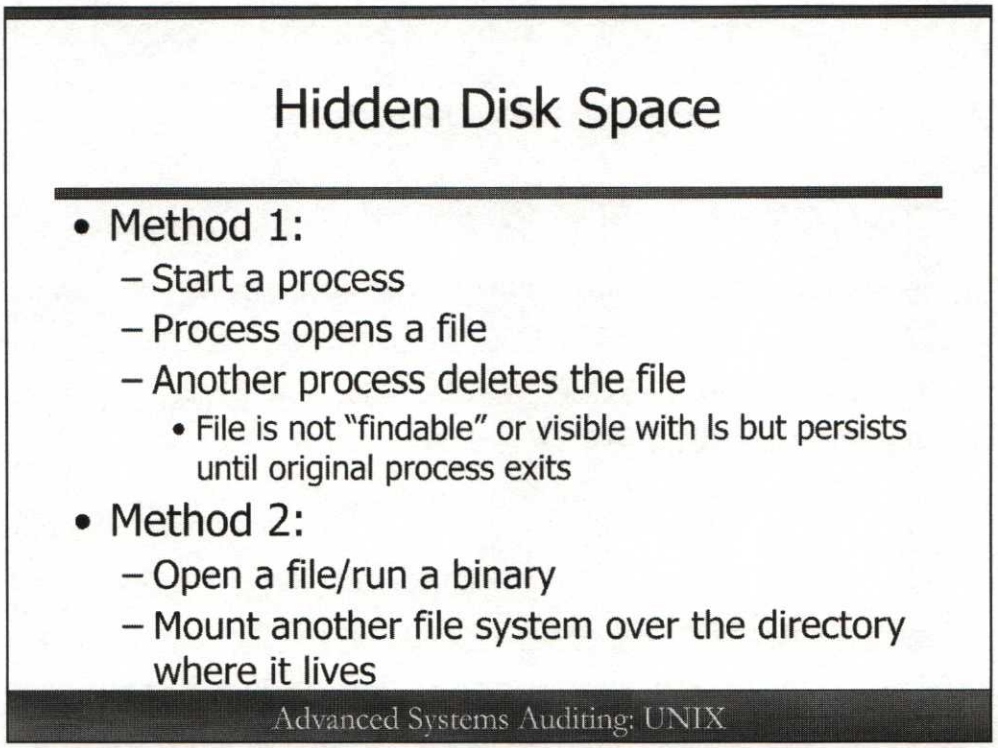
The touch command also allows you to set the MAC (modification, access, creation) times to any specific timestamp that you'd like to. In the slide, we set it to April 22, 2014. How did we come up with that date? Could it be that this was the last date that the system was audited? Is it possible that there have been anomalous log entries and we are looking for files changed since that activity was detected?

© 2016 David Hoelzer & Enclave Forensics™



Here, we can see 'touch' in use. Just touching a file will update its timestamps. Touching a file that does not exist will create a zero length file and update its timestamps, as seen above. Finally, using the '-t' option, we can set the Access and Modification timestamps directly. The format is CCYYMMDD[HHmm]. That's century, year, month, day, hour, and minute. The square brackets indicate that the hours and minutes are optional.

© 2016 David Hoelzer & Enclave Forensics™

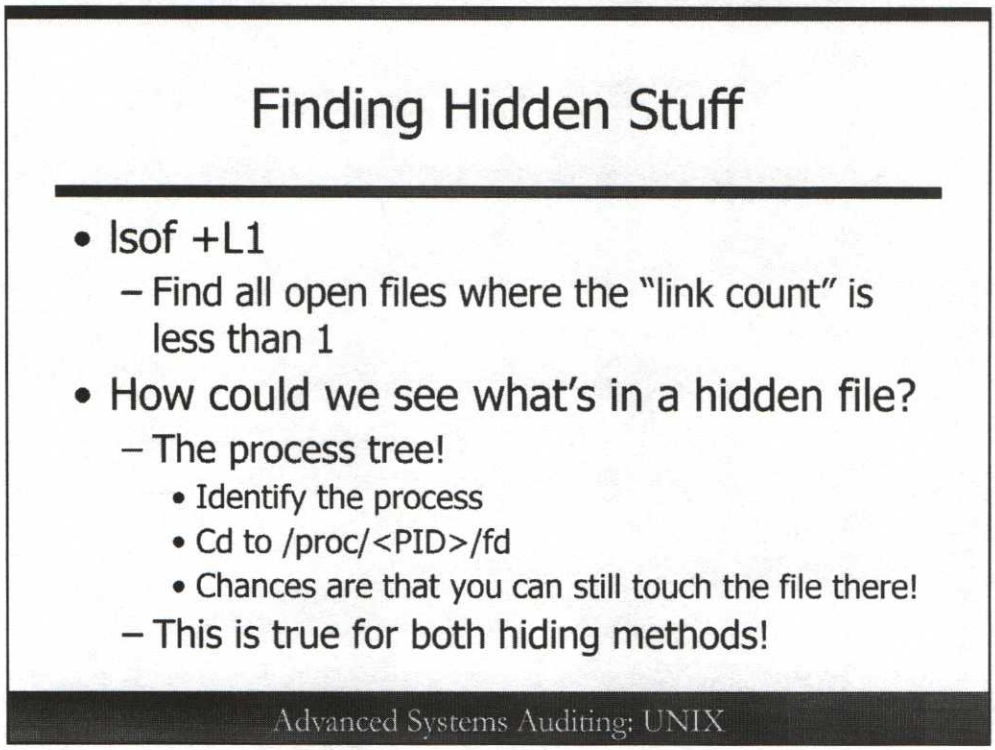


Hidden disk space in UNIX can be an interesting topic. We're not going to get into anti-forensics and other locations within disk structures and partition structures where files can be hidden, though if you're interested in this, you might browse this paper: http://www.berghel.net/publications/data_hiding/data_hiding.php On the other hand, we're not talking about something simple like prefixing a filename with a dot. Let's look at two somewhat advanced methods that are well within the capability of the average hacker or administrator.

First, start a process. This process opens a file. Meanwhile, another process deletes or "unlinks" the file. Because the first process has not closed the file, the disk resources will remain in use even though there is no longer any direct file system artifact to find. The reason is that the file, or inode, has been marked as deleted even though it is still open.

The second method is even more awesome. Create a trojaned binary. Execute the binary so that the process is up and running. Now mount another file system over the directory where that binary exists. For extra credit, put another innocuous binary with the same name in the mounted file system. The original file is unreachable using any normal means, nor can the administrator determine trivially that execution is not related to the file that is visible in the directory! The only way to find it is to unmount the file system and look again.

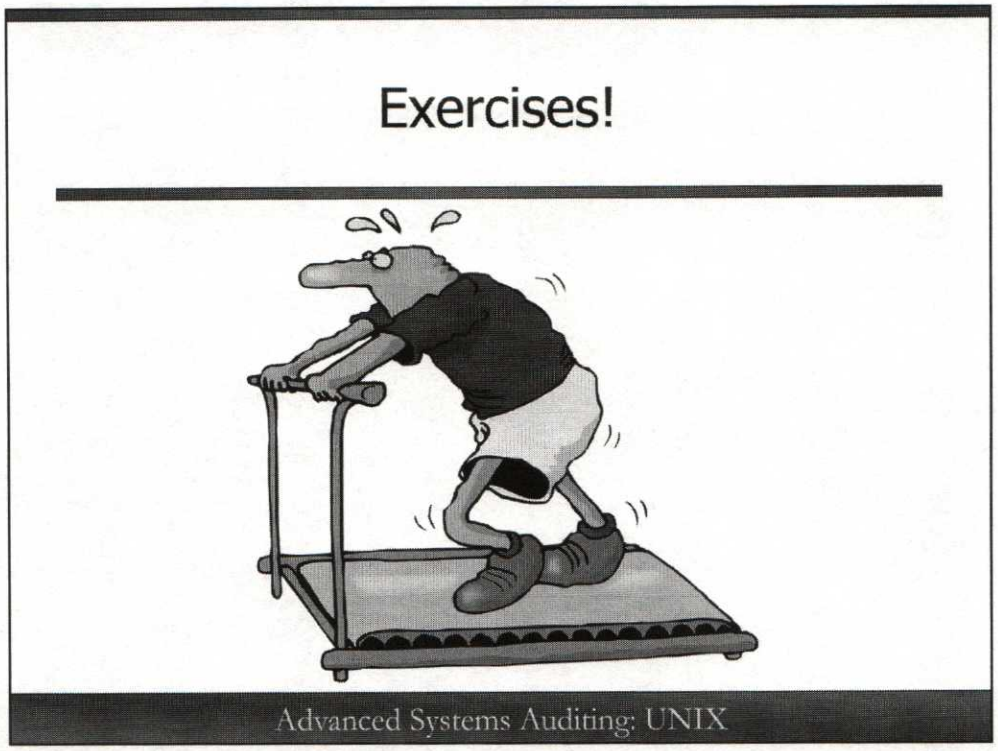
© 2016 David Hoelzer & Enclave Forensics™



Now that we know how to find hidden disk space, one of your immediate questions will be, "How can I see what's in that space?" The answer is, the proc file system! The proc file system is a relatively new development in UNIX. Remember that we said that UNIX tries to treat everything as a file? Well, the same is true of the processes! These days, the process space and kernel are available to use through the file system in the '/proc' file tree!

If you move into the /proc tree, you will see a bunch of directories with numbers for names. These numbers represent running processes. If we use lsof to identify hidden file space, and then identify which process is holding the file open, there is a reasonable chance that we can access the file through the proc file system even though it has been removed from the actual file system! We're not going to dig into this any more in this class because this is more of a forensics topic than an audit topic. If you want to try it out, change into the /proc directory, and then change into any directory representing a process. Within that directory, there will be a subdirectory named "fd" for File Descriptors. If you look in this directory, there will be a series of numbered files displayed that are linked to open files. Even if a file has been deleted, if the file is still open you can grab the file handle from here and copy or access the file!

© 2016 David Hoelzer & Enclave Forensics™



Please continue on with the exercises in the workbook from wherever you left off. Please feel free to experiment with some of the items discussed in class so far and ask a proctor or the instructor for help if you run into trouble!

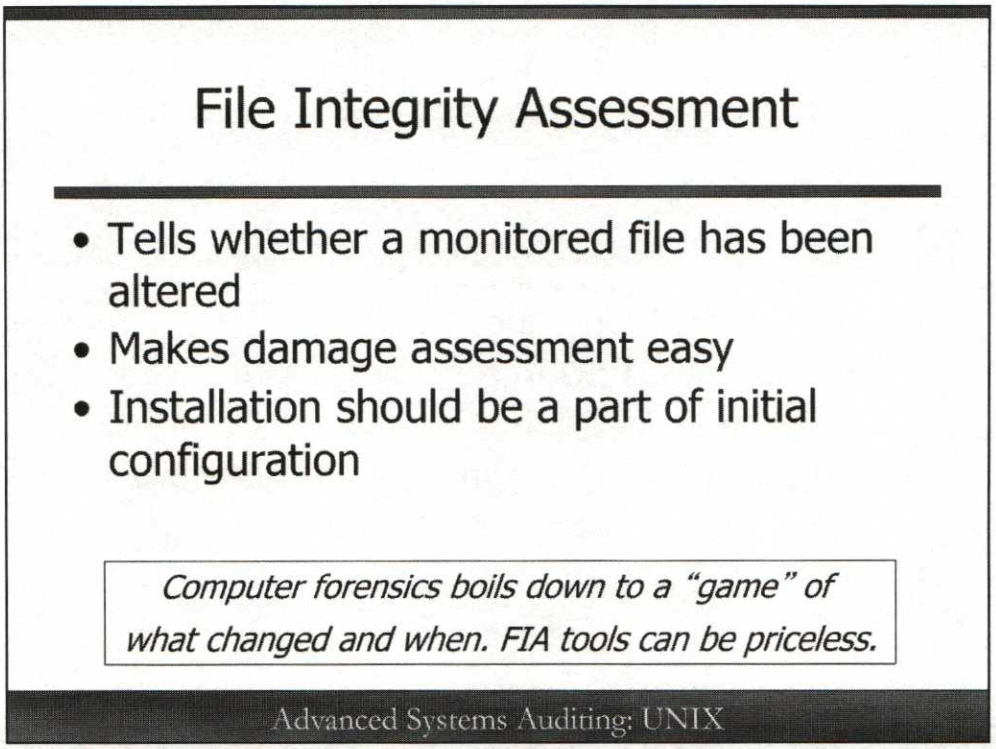
Audit Objective

- Objective: Unauthorized Access
  - Ensure that critical system files have not been modified
- Audit Activity:
  - Tripwire
  - File Integrity Assessment

Advanced Systems Auditing: UNIX

Identifying critical issues like file permissions and SUID files is important, especially for a baseline, but it isn't comprehensive. We also want a tool that will allow us to baseline and then identify changes to virtually any file on the system. This is where file integrity assessment tools come into play.

© 2016 David Hoelzer & Enclave Forensics™

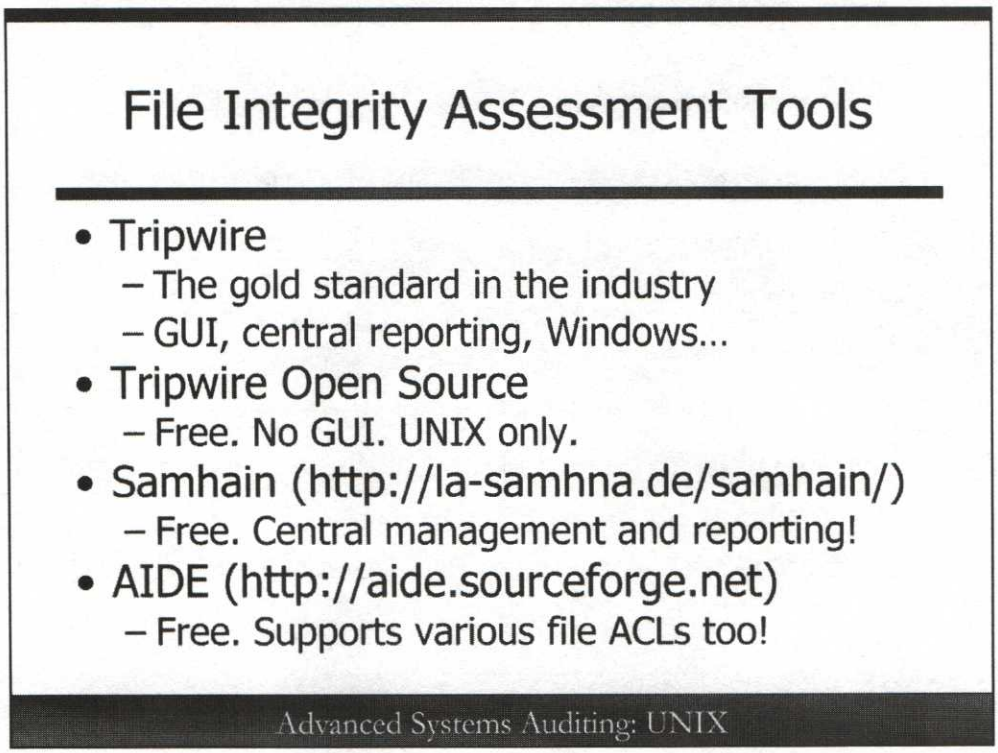


Make sure that any system deployed has the ability to detect changes in important system files. Something as simple as adding a "++" to the /.rhosts file allowed Kevin Mitnick to render Shimomura's system completely defenseless.

Of course, being able to detect that a file has changed means that the system was compromised, and that's kind of late in the game. But we would still rather know 12 hours after the compromise, than not to know at all.

The ability to detect changes in the files of the file system also makes recovering from an incident much easier. In this case, you often are able to assess exactly which files were changed by the attacker. This means that recovery can consist of replacing the hacked files (and securing the system) as opposed to a complete reinstall and restore from backups.

© 2016 David Hoelzer & Enclave Forensics™



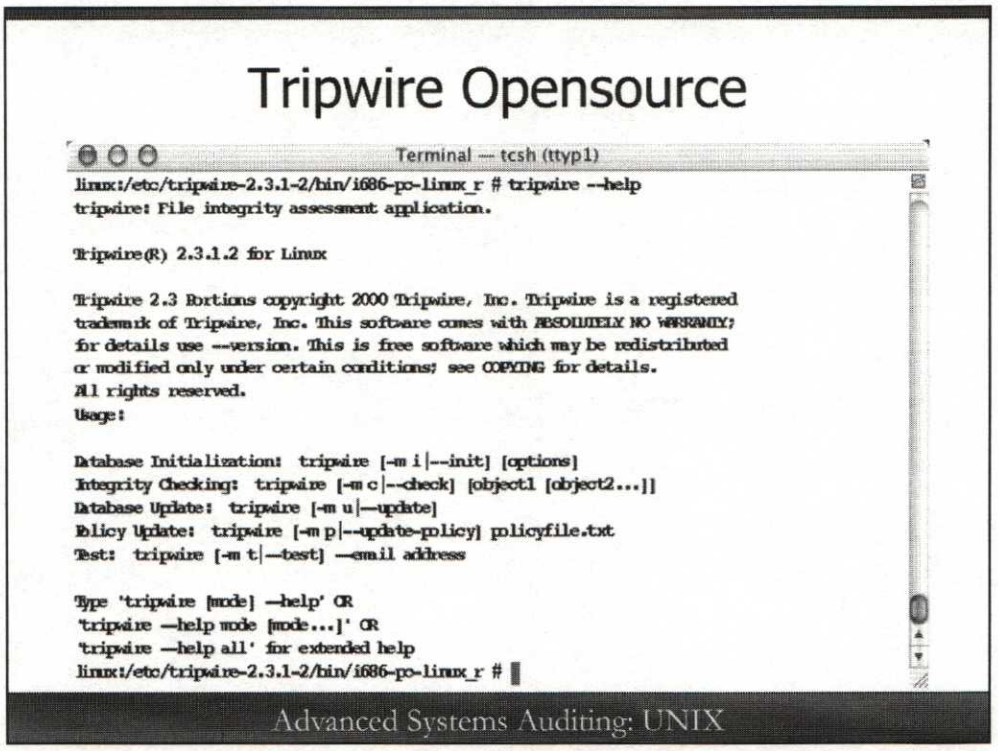
The installation of file integrity-testing software is an absolute must for any server class system. The installation of this software should really be a part of the base configuration of the system.

There are lots of options in this market space for UNIX. You should push your organization toward Tripwire because it will allow you to create a single file integrity management infrastructures across your enterprise. Unfortunately, that's going to cost you some money.

If money is tight, you might start by looking at the open source version of Tripwire. We look at an item or two that can cause headaches for administrators when first using this tool in a second.

Two other alternatives are Samhain and Aide. Samhain is particularly interesting because it provides central management and reporting as well as a web-based interface for the reporting. There are even examples where some have convinced Samhain to work on Windows hosts! Aide is also interesting because it provides additional support for non-POSIX file access control lists.

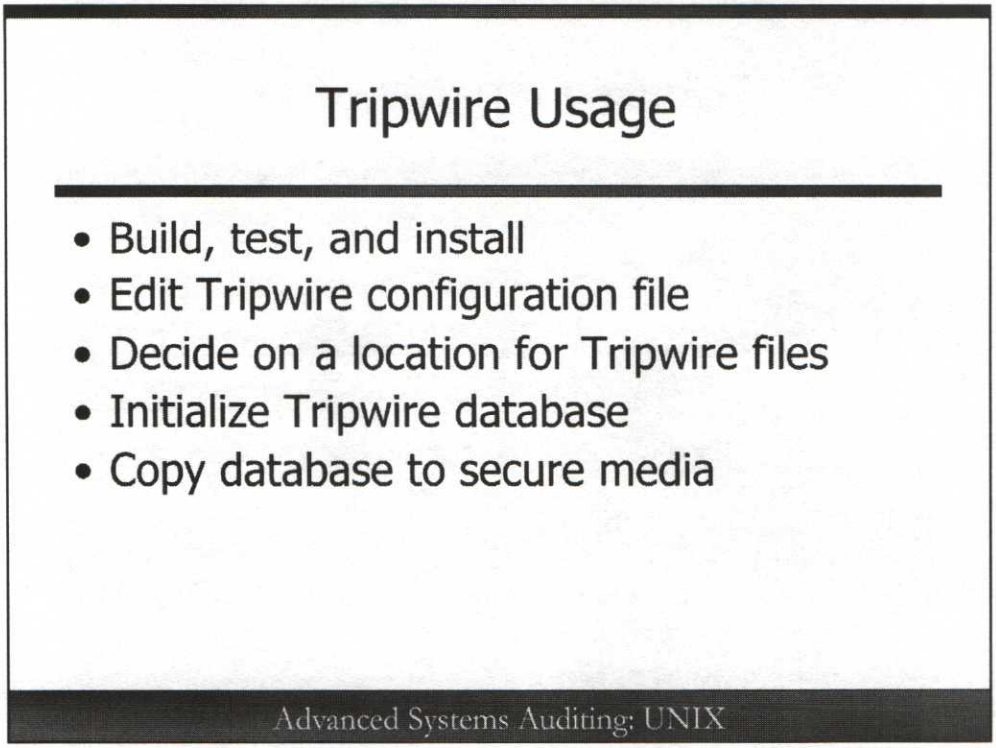
© 2016 David Hoelzer & Enclave Forensics™



In the slide, we are picturing the open source version of Tripwire. This version has been written with Linux specifically in mind, though it will work on Solaris or BSD with a minimal amount of effort. The open source version is available from http://sourceforge.net/projects/tripwire/ and is also free for use by the security community.

With all of these free options, you might wonder why you would want to purchase the commercial version of Tripwire! The answer is that the commercial version includes some fantastic reporting options in addition to being available for multiple platforms that are not supported on the opensource side. These multi-platform versions also work cooperatively, giving you the opportunity to create integrated reports.

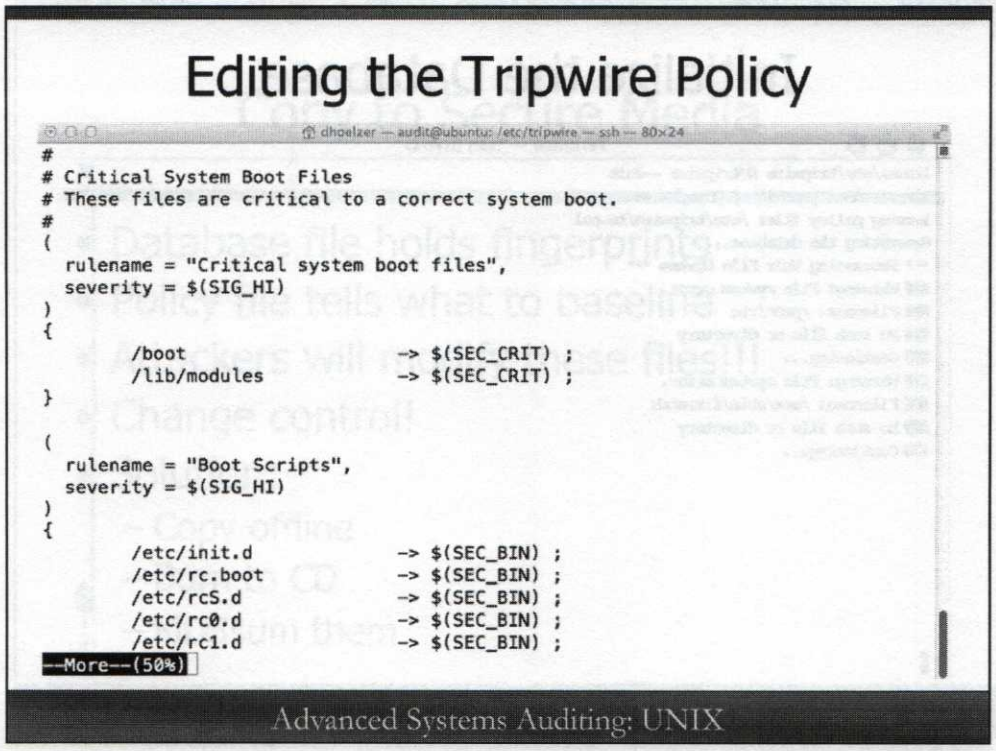
© 2016 David Hoelzer & Enclave Forensics™



To put Tripwire into use, there are a few steps that you need to take. First, after deciding on which version you plan to use, you must build the source code into binaries. The Tripwire source code, if you choose to use the Academic Source Release (ASR) or the OpenSource versions, comes with detailed instructions on how to go about building Tripwire. If you purchase the Commercial version, you will not need to take these steps. You will simply need to install the software.

After the software is installed, you must edit the tripwire configuration file to customize Tripwire for your system and decide what you would like to baseline. There are a number of sample configuration files that come with Tripwire to get you started. Next, you decide on a location for the tripwire database and configuration files and copy them to this location. At this point, you are ready to go so you initialize the Tripwire database. This will perform an initial assessment of the system and generate fingerprints for everything specified in the configuration file. Finally, you must move the database that is created to secure media to prevent tampering.

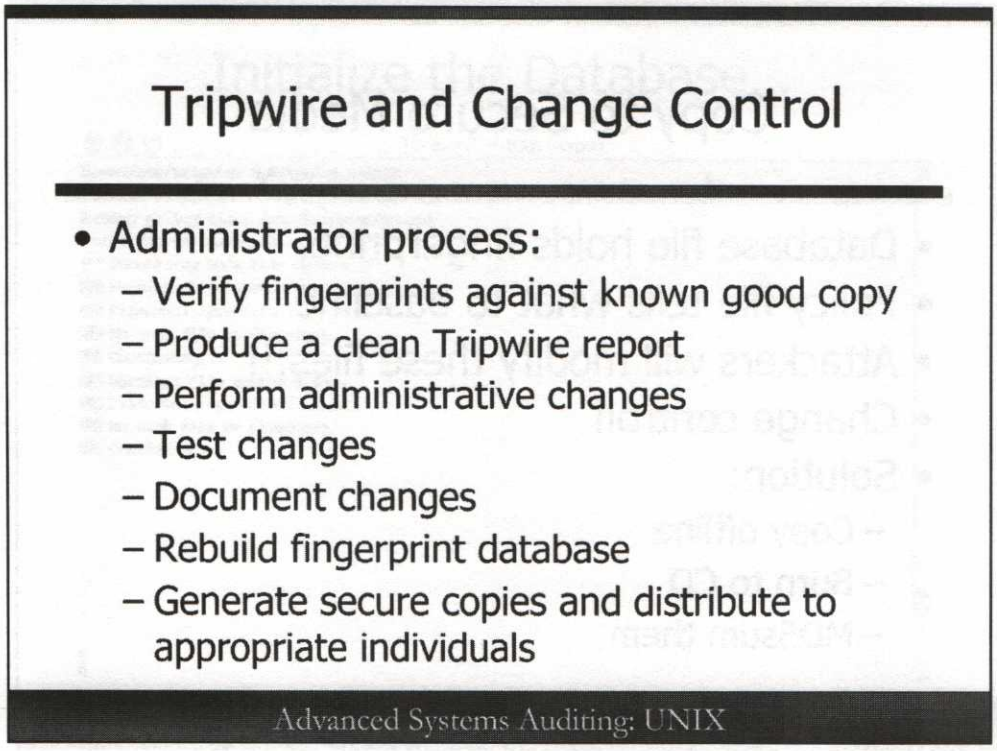
© 2016 David Hoelzer & Enclave Forensics™



Editing the Tripwire policy file is pretty simple once you understand the format. We're not going to spend time discussing each of the individual configuration options for the policy file. The key is to identify portions of the system that should and should not change and enumerate them in this file. After this is done, you can create a pretty good baseline of the system looking for changes in everything from modification times and sizes all the way to the inode used by a file.

The administrator really ought to go over this file carefully before initializing the database, but there are thousands of files on any given UNIX system. More typically, as mentioned on the last page, he will run the initialization and then come and clean it up by editing this file. He's headed for some frustration, though, and we just want to prepare you for that. If you are introducing this tool to an administrator, it would be a really good idea to explain what's going to happen next…

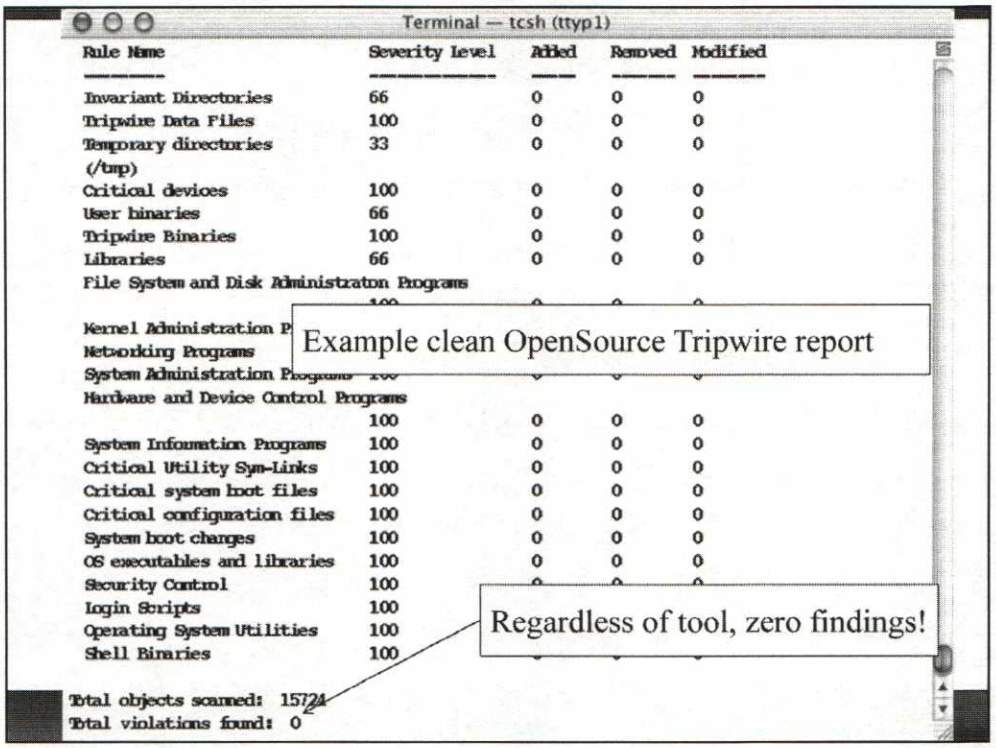
© 2016 David Hoelzer & Enclave Forensics™



The process that your administrators should follow is a pretty simple one and does not require an onerous amount of labor. In fact, it quite likely incorporates security procedures that are not followed today! The idea is that the administrator must first verify that the system has not changed since the last baseline was run by checking the file fingerprints against the last known good copy of the database. After this is completed, the administrator can go ahead and do whatever patching or installation work needs to be done and document those changes appropriately. The next step is to rebuild the fingerprint database and to create secure copies of it to be distributed to the responsible individuals.

In an environment with hundreds (or more) of servers, the process would likely not send a copy of the database to the auditor after every change, but there would be secure copies kept in the datacenter vault, for instance. The auditor will always have his own last known good baseline and can verify change control process documentation by comparing the report from his known good copy against the running system, and then simply update his own baseline.

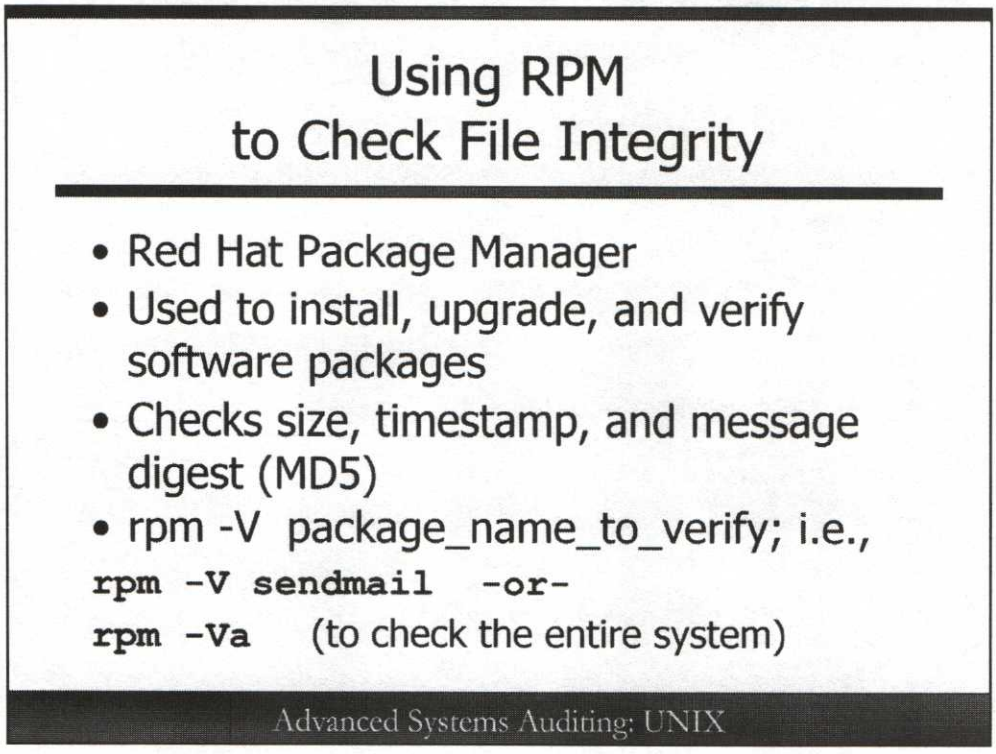
© 2016 David Hoelzer & Enclave Forensics™



This is a run of the free tripwire OpenSource. It runs on Linux out of the box and most other Unices with minimal effort. Note that in this example, we are scanning our system for any changes. In this case, we ran the audit with the command line 'tripwire –check' to find exceptions. All in all, this scan evaluated over fifteen thousand files on the system and took about three minutes to run.

There's a very important point here. At the bottom of this report, we can see that the total number of violations found is zero. There is no good reason why there should ever be any exceptions. In fact, if you see a number other than zero, let me translate what the administrator is saying to you. He will begin his sentence with, "Oh, that's expected. That's because…" We really don't care what words come next. Whatever he says, his words can be translated to, "That's because I haven't configured the tool correctly." For this tool's output to be meaningful, it must be properly configured! Without that, it's not worth installing it!

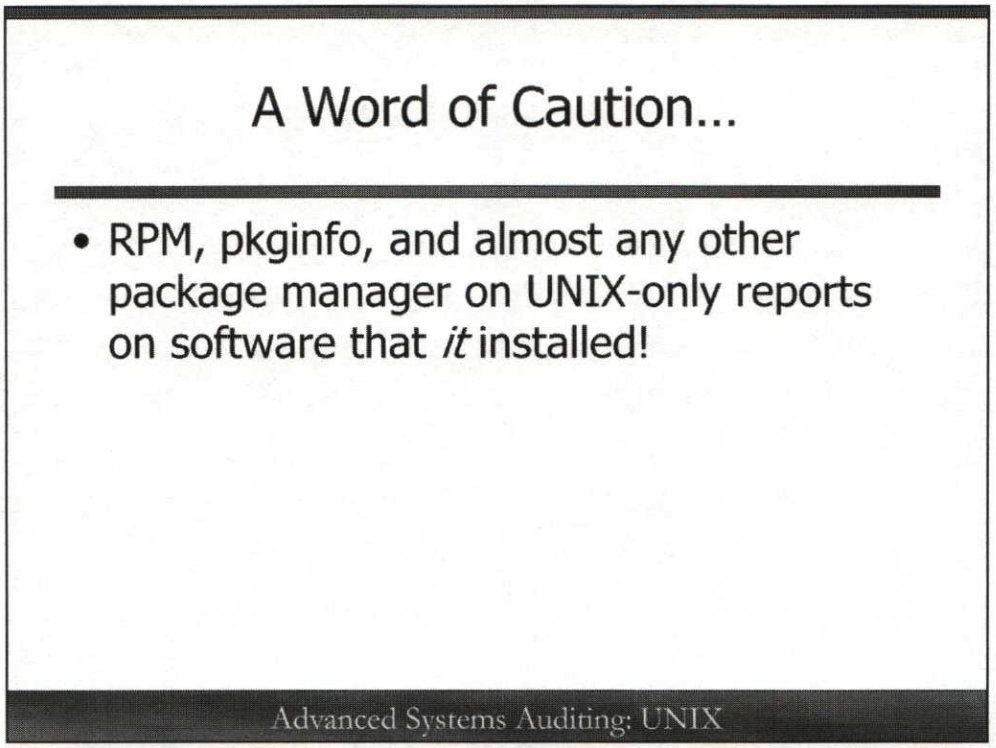
© 2016 David Hoelzer & Enclave Forensics™



If you begin the conversation about file integrity testing with the administrator, be prepared for another objection. He might say, "Oh, yes, I know about that. But we don't need it because we're using [X] instead!" The X here is some type of package management product like Apt, Yast, RPM, or some other similar installation tool.

Although these tools have done a great deal to simplify administration of UNIX systems and the installation of software, they are *not* file integrity-testing tools! They certainly have some hashing features built in, but the purpose of these is to simply inspect installed software for the purpose of repairing installations or applying patches.

© 2016 David Hoelzer & Enclave Forensics™



Although these tools are valuable, they are capable only of reporting on the integrity of files that they themselves installed! They will not be able to tell you whether there are new files or whether there were files modified that were not a part of a package installed or maintained through the package manager that you are using!

This is a pretty important distinction between a tool like Tripwire and the RPM package manager. If you want to verify a package, by all means use RPM. If you are trying to verify a system, Tripwire (or something like it) is the correct tool for the job.

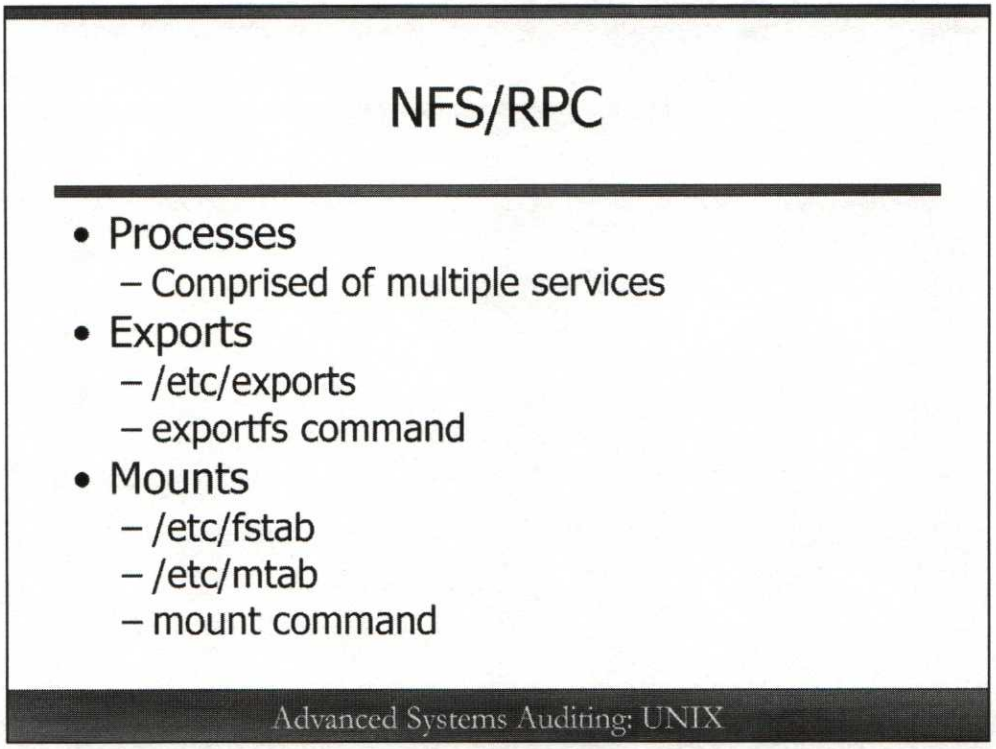
Audit Objective

- Objective: Operating Profile
  - Identify potential information disclosures
- Audit Activities:
  - Login banners
  - Service banners
  - File permissions
  - NFS services

Advanced Systems Auditing: UNIX

The next step in our operating system audit process looks at other operating profile exposures and operating system specific vulnerabilities. On UNIX systems, this includes information disclosures (like banners and weak file share permissions) and commonly enabled unnecessary services (like NFS and other RPC programs).

© 2016 David Hoelzer & Enclave Forensics™

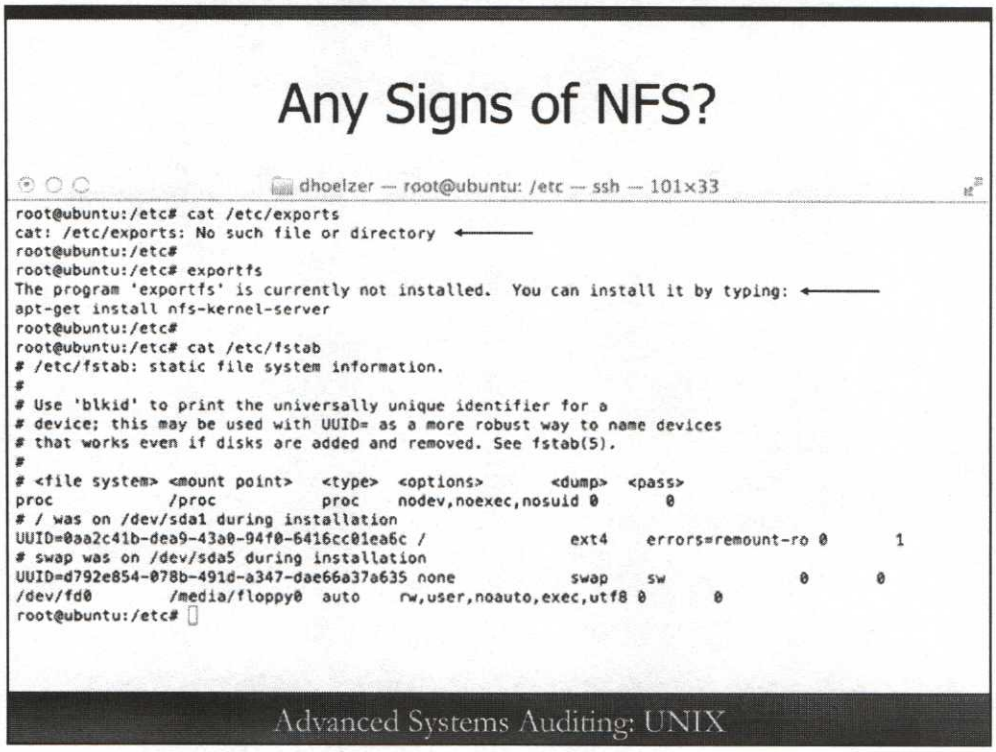


A top issue for both Windows and UNIX concerns weak or non-existent file-share permissions. Although it is absolutely true that a UNIX system could be using SAMBA to create Windows styles shares, in our experience this isn't where the problems are. If an administrator is using SAMBA, he is usually using it to connect his UNIX system to a domain share to access files, not to share folders to the Windows network. This isn't to say that it can't happen, though, so if we found SAMBA running we should certainly inquire and check on the file share permissions.

In our experience, the weak file share permission issue in UNIX typically involves NFS. We see UNIX systems used as a sort of network "duct tape" to connect two disparate systems, allowing for data to be transferred. For example, in one enterprise, we helped it to create a transport between its brand-new ERP system and its legacy shipping system. What was the transport?—a UNIX system.

"Pick tickets" were generated on the ERP system and dropped into an NFS share. The UNIX system would pick them up, process them, and then share them with the shipping system. Why NFS? Because it's the native file-sharing application for UNIX. It is an Remote Procedure Call (RPC) service made up of a number of services like nfsd, mountd, statd, and others. It typically has a configuration file called "/etc/exports" and will often create entries in the /etc/fstab for frequently mounted NFS volumes.

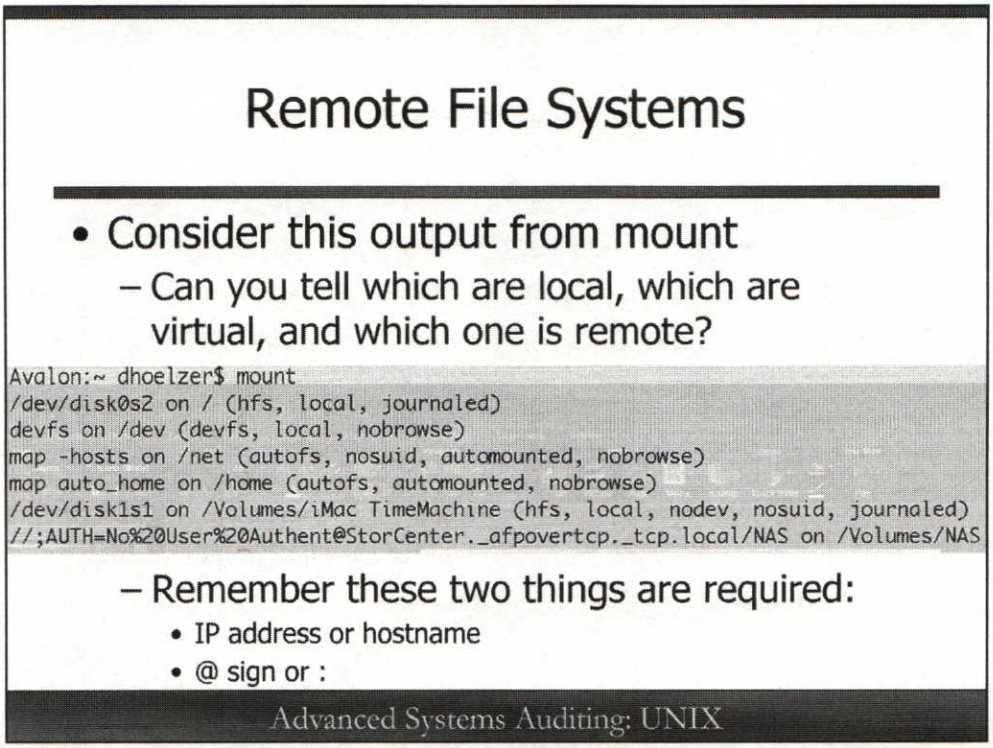
Although we're dealing with an RPC (NFS is an RPC service), let's talk about RPCs in general. A Remote Procedure Call (RPC) is a well-defined system for registering available services that are essentially published, or at least available, on the network. We can query a system to determine which RPC services are available by running the command 'rpcinfo –p.' If the system has any RPCs installed, this tool will almost always be installed. We can also install this tool on a system, and then query remote systems. When running this tool, the results will reveal any RPC Programs (RPCs are identified by unique program numbers, not by port numbers) that are available on the system. These should be minimized and should absolutely be a part of any good baseline of the system.



Now that we're aware of what NFS is and how it works at a high level, we can look for signs of it. In this case, we checked for the /etc/exports file, which apparently doesn't exist—a promising sign! We also tried to execute the 'exportfs' command, which is used to create NFS shares. This isn't installed. Lastly, we look at the /etc/fstab (file system table). This is a configuration file where an administrator will create entries for all commonly mounted file systems. We are looking for hostnames, IP addresses, and other things that look like remote systems. In this case, we find only local devices in the fstab.

In this case, there's no sign of an NFS share. How hard is it to identify a network share just by sight?

© 2016 David Hoelzer & Enclave Forensics™



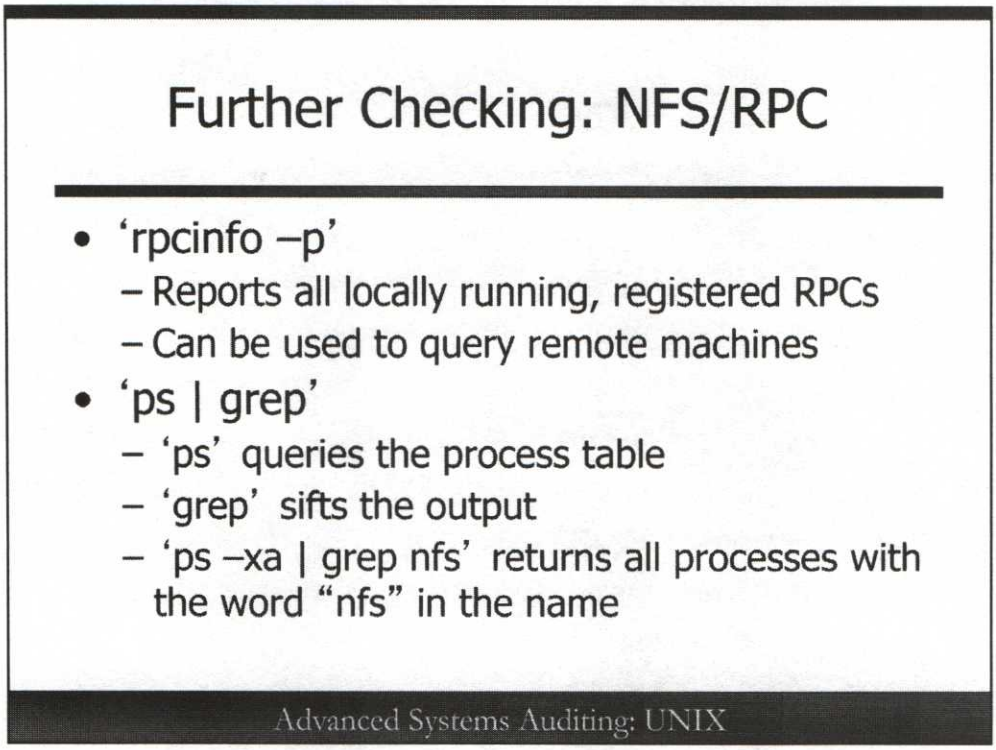
When looking at the fstab, the mtab, or the output from mount, you are looking for file systems whose device has an @ or a : in it with a hostname or IP address. Look at the output of mount in the slide above. Which of those file systems are local? Which ones are virtual file systems? Which of these is a remote file system?

Notice that two of the lines begin with "/dev". That's a dead giveaway that these are physical drives. Two of them begin with the word "map". Now, that's not a requirement, but it's not a hostname or an actual device. For example, on Linux systems, you'll find "/proc" there. This still is pretty clearly not a device. These are virtual file systems of some kind.

Notice the last entry, though. This one has some kind of Auth data, followed by an @ sign, and then a system name on the local network. This is a remote file share. Whether this is an SMB share, an AFP share, an NFS share, or some other kind of file share would appear to the right. We have intentionally trimmed that out of this example. The point is that identifying remote file systems really is trivial!

This gives us a very simple rule of thumb. If a remote file system is being mounted, we should see an entry that has either an IP address or a hostname in it. That same entry should also have an @ sign or a : that is used to separate the credentials and/or hostname from the actual path.

© 2016 David Hoelzer & Enclave Forensics™



On the local host itself, there are some other tricks we can do to determine whether there are RPC programs running or whether NFS is running. The easiest is to simply use the 'rpcinfo' utility. This program is used to query the portmapper directly and will return, among other things, a complete list of all currently running and registered RPC programs. We can even run this against a remote host by entering something like:

```
rpcinfo –p remoterpcs.mydomain.com
```

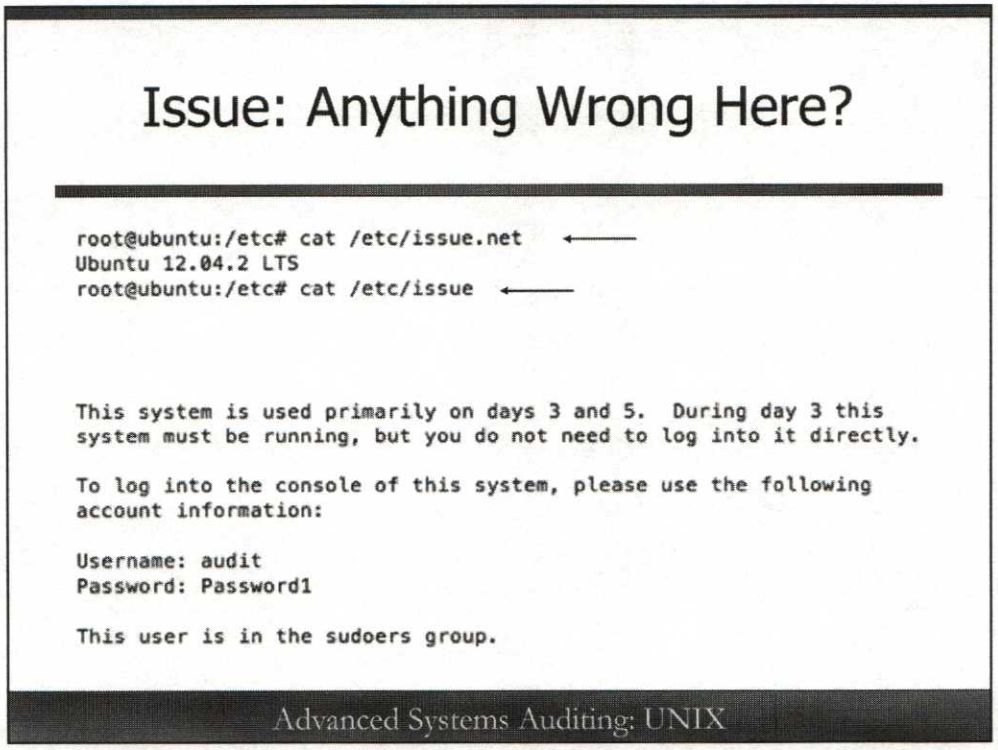
It is possible, however, for an RPC program to be running but not registered with the portmapper. The most common way that this happens is that either the portmapper is restarted at some point or the RPC service was started before the portmapper was started. In either of these cases, the portmapper will likely not have the information we seek. In fact, the portmapper might not even be running!

To find these services under these circumstances will require that we have done the research necessary to audit the particular brand of UNIX that we are looking at. We will search the process table using the 'ps' command and try to identify programs that could possibly be RPC programs. For instance, we know that nfsd, statd, lockd, and mountd are all related to NFS. We can search for them like this:

```
ps –eax | grep '(nfs)|(lockd)|(statd)|(mountd)|(rpc)'
```

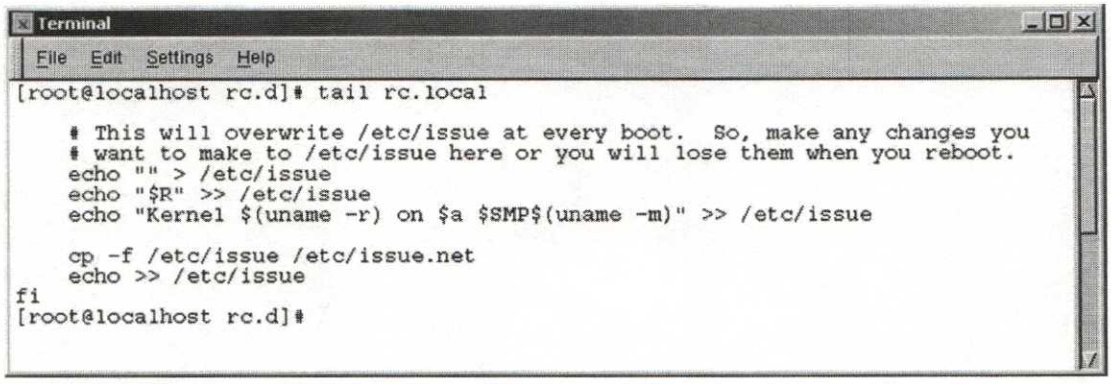
Notice that we added the letters "rpc" to the end. On many systems, RPC programs will actually begin with "rpc.", allowing us to identify them more easily using this method.

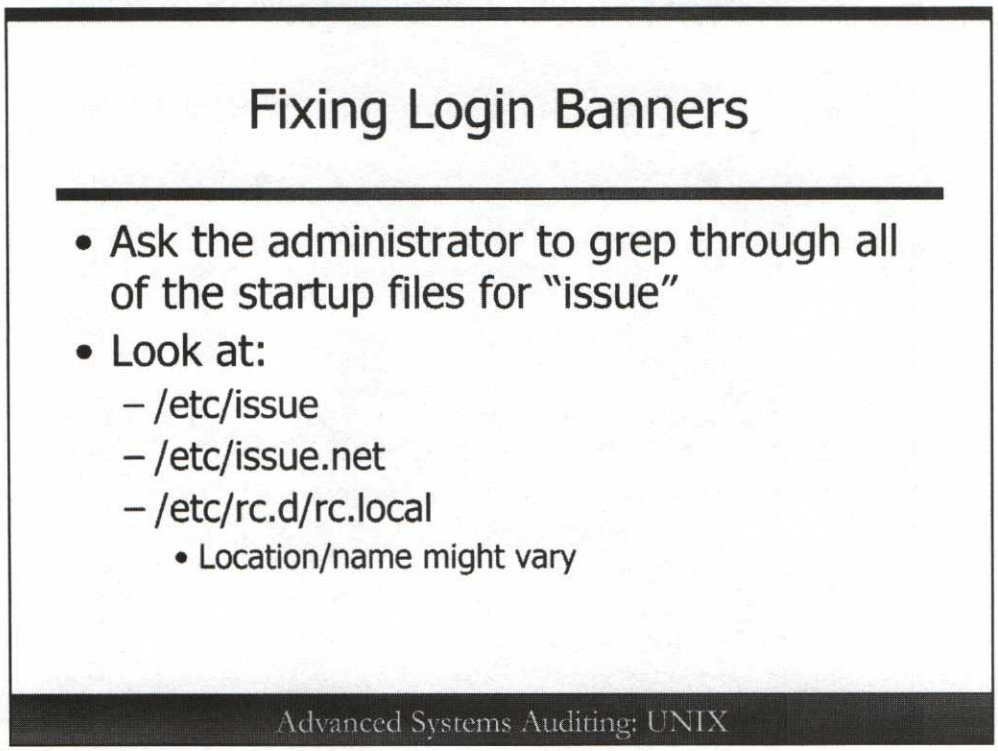
© 2016 David Hoelzer & Enclave Forensics™



Another form of information disclosure can be found in the /etc/issue and /etc/issue.net files. These files typically are rebuilt when the system is rebooted and populated with system configuration and build version information. This information is wonderful for an attacker because he simply needs to search in Google for matching vulnerabilities!

Be sure to check for any possible banner file. In our slide, you can see that there are actually two separate banners. One of these is displayed locally at the console (/etc/issue) whereas the other is displayed to remote users connecting to the system (/etc/issue.net)! Although the local console banner has been changed, the remote banner has not.





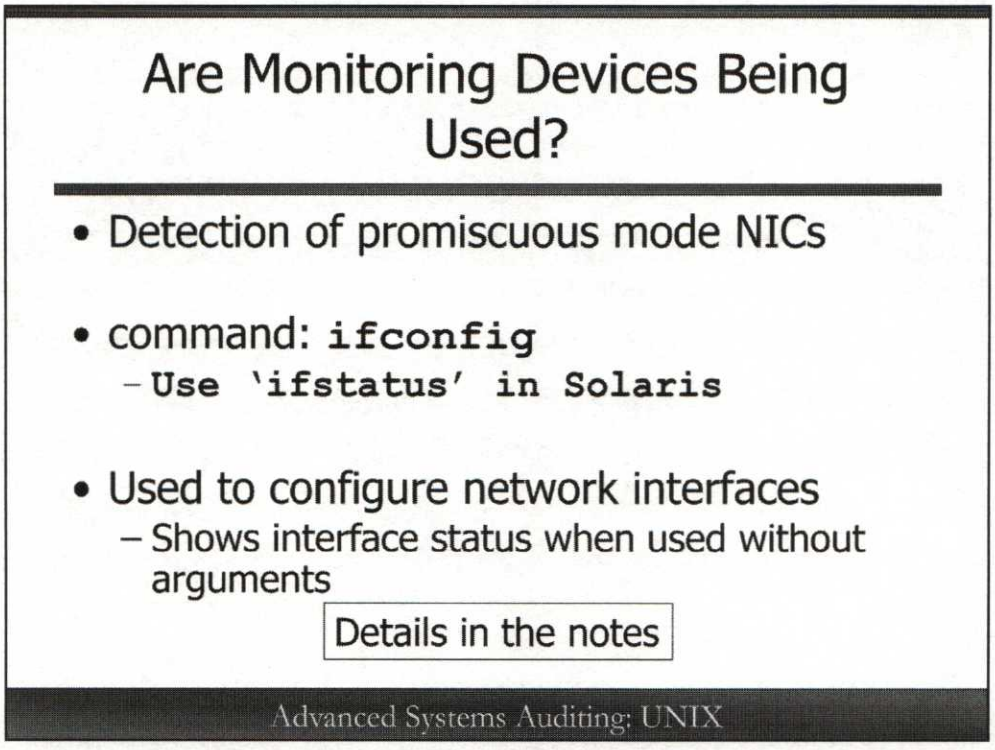
In the slide, we have documented some of the common places that you would find the banner files as well as one of the common locations where that file might be updated. It is a very good idea to ask the administrator to search through all of the startup files to ensure that these banners will not be replaced or regenerated when the system is rebooted.

From the /etc directory on an Ubuntu system, for example, he could execute a command like this:

grep –r issue /etc/rc*

This instructs the system to search for the word "issue" recursively in all files and directories whose names begin "/etc/rc", likely covering all of our startup scripts.

© 2016 David Hoelzer & Enclave Forensics™



One of the things that is usually found in a rootkit is a sniffer. They are primarily used to collect more username/password pairs. Network interface cards in promiscuous mode indicate that a sniffer is running. You will find that Solaris does not always report accurately when using the ifconfig tool. A more reliable tool is 'ifstatus.'

```
[root@localhost /root]# ifconfig
eth0 Link encap:Ethernet  HWaddr 00:E0:98:7D:1F:AC
     inet addr:192.168.1.5 Bcast:192.168.1.255 Mask:255.255.255.0
     UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
     RX packets:15 errors:0 dropped:0 overruns:0 frame:0
     TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
     Interrupt:5 Base address:0x300

lo   Link encap:Local Loopback
     inet addr:127.0.0.1  Mask:255.0.0.0
     UP LOOPBACK RUNNING  MTU:3924  Metric:1
     RX packets:0 errors:0 dropped:0 overruns:0 frame:0
     TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
```

© 2016 David Hoelzer & Enclave Forensics™

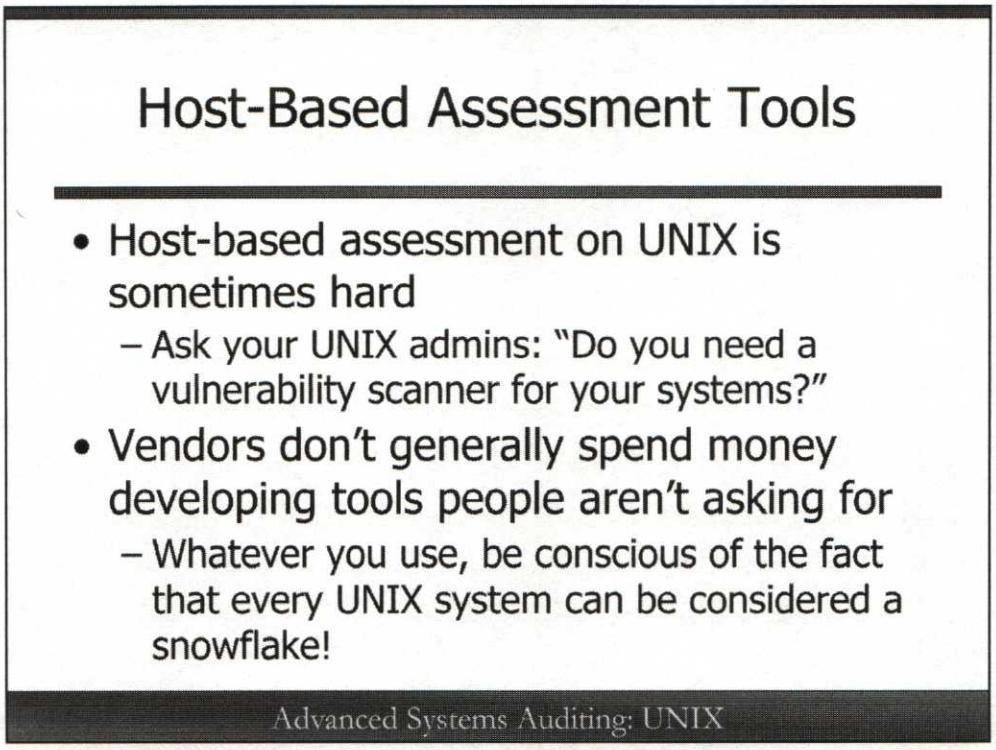
Audit Objective

- Objective: Secure Configuration
  - Test overall configuration for security
- Audit Activities:
  - Host-based assessment tools
  - Network-based assessment tools

Advanced Systems Auditing: UNIX

At this point, we have touched on nearly all of the major areas that should be audited on our UNIX systems. One last set of testing is recommended though; run a host-based and a network-based security evaluation tool against your system!

© 2016 David Hoelzer & Enclave Forensics™

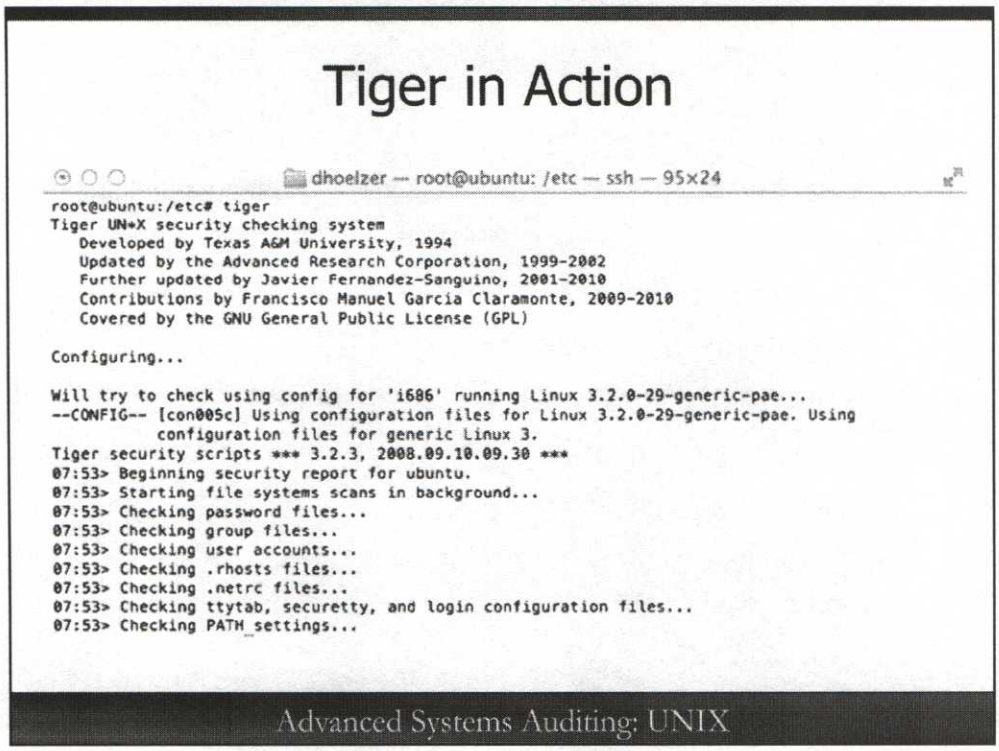


Host-based vulnerability assessment in UNIX is a more difficult problem than it is in the Windows world. One of the primary reasons for this is that UNIX administrators tend to strongly believe that their systems are more secure than Windows hosts. In fact, if management were to ask them, "Hey, would it be useful for you if we bought a vulnerability scanner for your UNIX systems?" the administrators will often answer, "Oh, we're fine… you should go talk to the Windows administrators!"

Frankly, UNIX systems are equally likely to have vulnerabilities discovered. This attitude, however, means that there is very little effort or money spent by vendors on developing really solid UNIX vulnerability scanners. UNIX support tends to be more restricted (Solaris and Linux, for example, but little to no real support for AIX, Dynix, Irix, HPUX, NetBSD, etc.). Additionally, recall that we said that even from version to version, files will absolutely move around on UNIX systems. The administrator has a big impact on this too! Some administrators tend to "Nest…" They, essentially, move things around to where they *like* them to be. As you can imagine, this makes vulnerability scanning very challenging!

What this means is that, regardless of which tool you choose to use, you must look very carefully at the configuration of the tool. You must also look closely at the log file that it generates rather than just relying on the report that is created!

© 2016 David Hoelzer & Enclave Forensics™

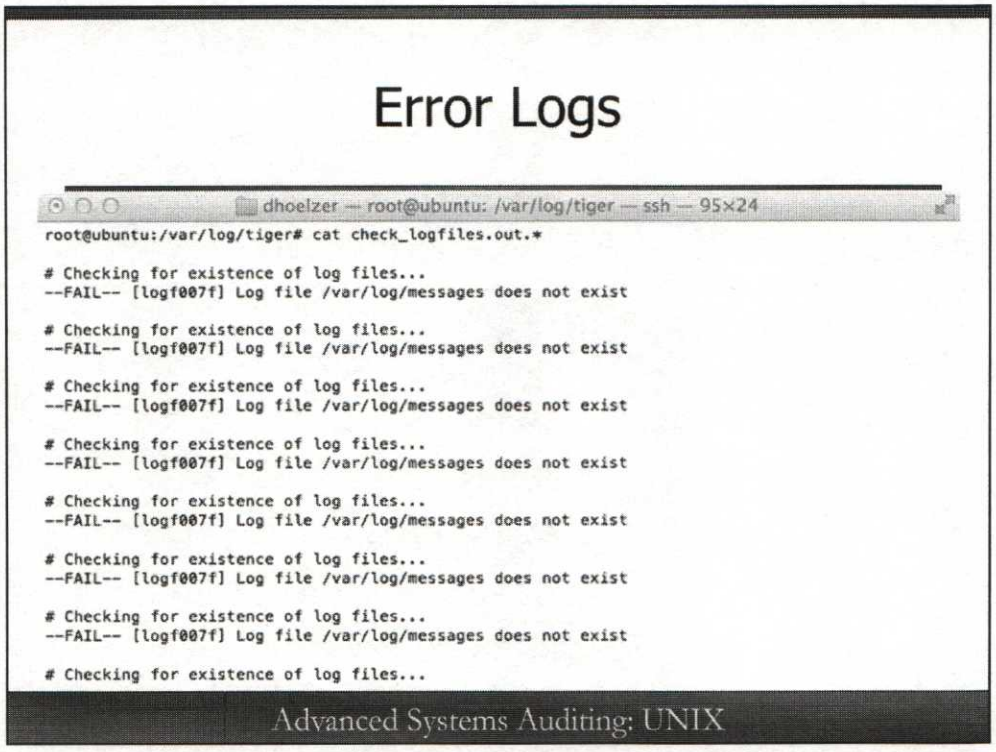


Here, we can see Tiger, one of the modern-day replacements for COPS, in action. As it's running, things look pretty good.

There's an extremely important thing to know about vulnerability assessment on UNIX. It's hard. The trouble is that every UNIX system is different, software is installed differently, configuration files could be anywhere… It's just hard. Worse, there's little to no market for UNIX vulnerability scanning tools. This means that companies just aren't investing the kind of dollars into development that they are for Windows vulnerability scanners.

As a result of this, there is an extremely important thing that must be done whenever performing vulnerability analysis on a UNIX system… Have a look at the next slide.

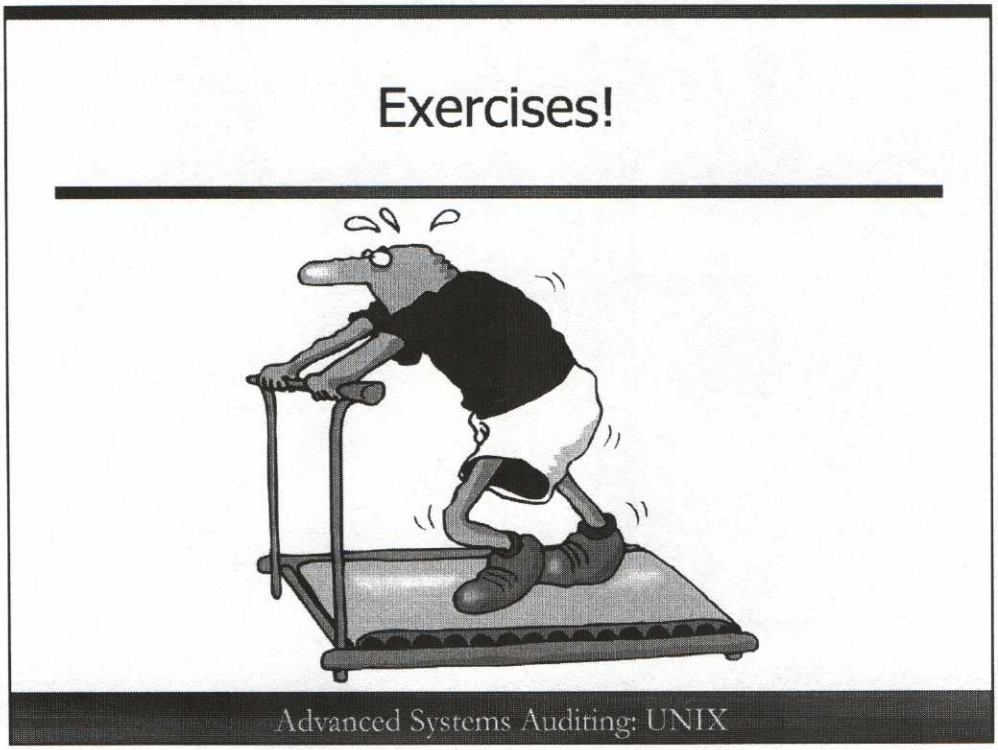
© 2016 David Hoelzer & Enclave Forensics™



You absolutely must check the error logs generated by the vulnerability testing tool. In this case, we can see that Tiger was trying to run tests against the system log. Unfortunately, it couldn't find the system log. Is that because there isn't one? Isn't it far more likely that the administrator has simply moved the log file somewhere else, and Tiger has failed to find it.

So, then, even if we get a completely clean *report*, we must look at the log files that the tool creates to verify that the tool was properly configured. If we don't, we could end up with false negatives—a very vulnerable system that the scanner things is okay.

© 2016 David Hoelzer & Enclave Forensics™

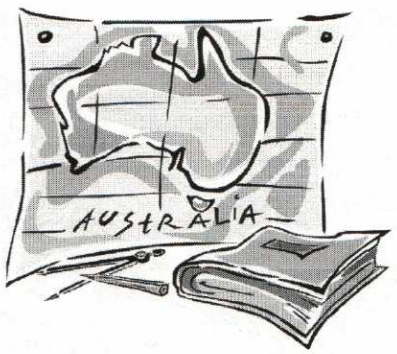


At this point, please continue on in your exercises book. You should be in a position to complete all of the remaining exercises. The instructor will call you back sometime in the afternoon to complete the last 11 slides of the presentation.

© 2016 David Hoelzer & Enclave Forensics™

Roadmap

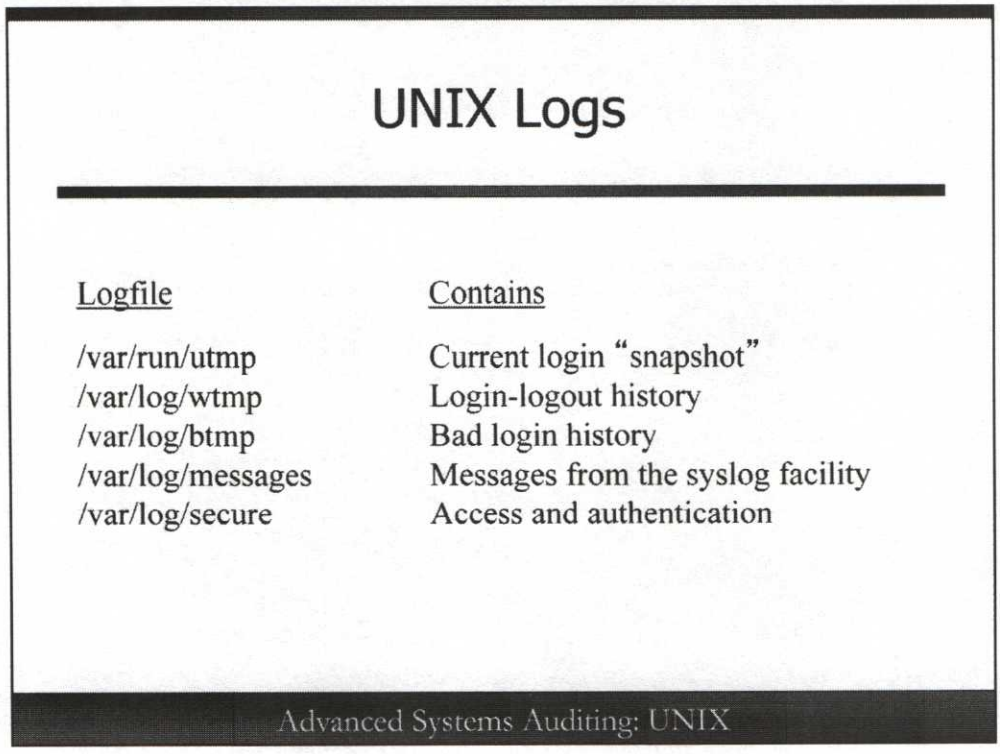
- UNIX Logging
  - Messages
  - Syslog
  - Utmp
  - wtmp



Advanced Systems Auditing: UNIX

Before we get to actually evaluating a UNIX system and performing audit actions, we first need to identify the core audit trails that exist by default, see what's in them, and determine where they can be found. For the next few slides, we will discuss UNIX logging features.

© 2016 David Hoelzer & Enclave Forensics™



The various log files of the UNIX operating system are often referred to as the "audit logs." So, it's no wonder that as you poke around on the host in question, you'll want to check that all of the "important events" are logged to the appropriate places. As you'll see, sometimes that involves making sure that the appropriate places exist in the first place. The following window shows a listing of the log files that are written in the /var/log directory. The auditor should examine each of these files because they should all have their own story to tell.

There are five major log files that the administrator should keep a watchful eye on for signs of unauthorized activity. For the most part, these files can be found in the /var/log directory of the Linux file system. The notable exception is utmp, which can be found in /var/run.

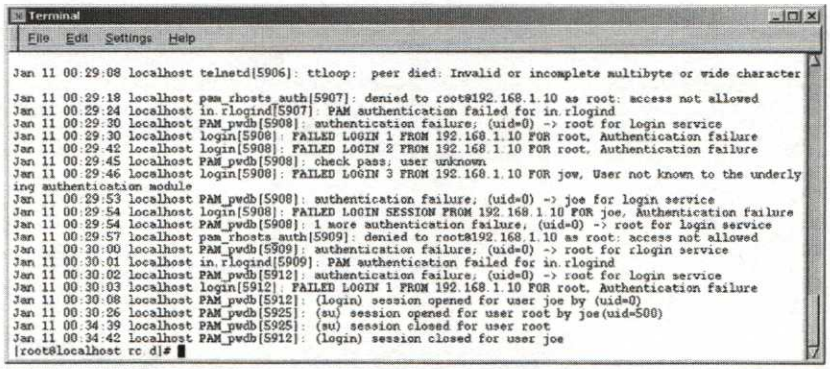
It is important to understand the contents of each of these log files and how one can exploit the information to achieve a more secure system. Starting with the next slide titled, "The messages File," we discuss the details of each.

© 2016 David Hoelzer & Enclave Forensics™

'messages' Log

A copy of each system message that is displayed on the console is stored in the **/var/log/messages** log file

The messages file is a very rich source of information

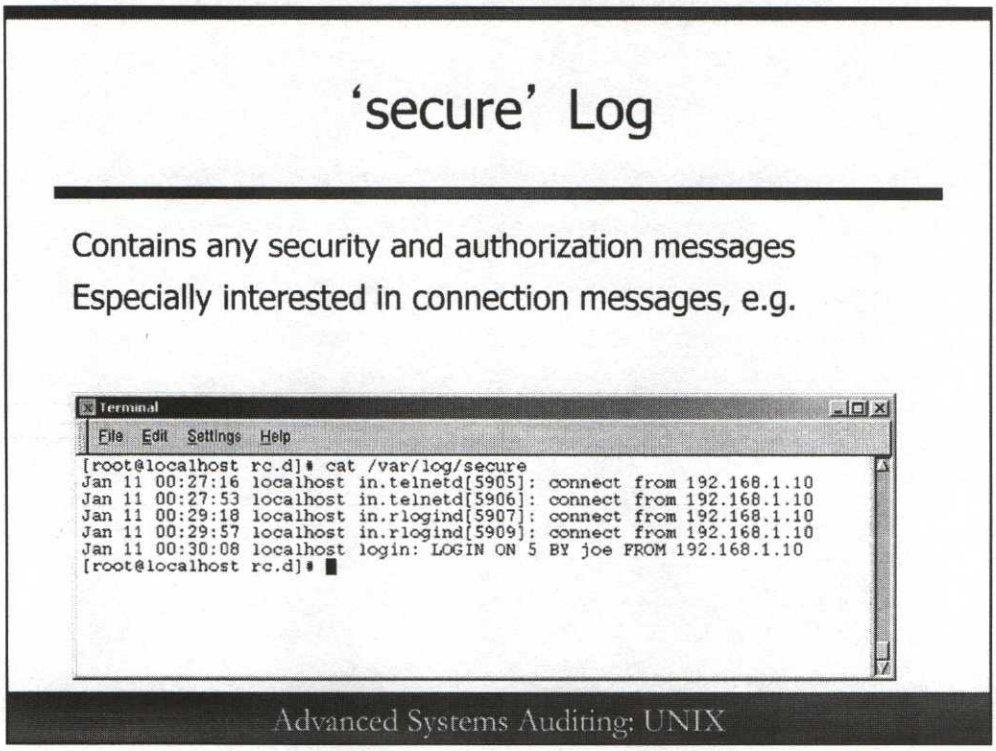


Advanced Systems Auditing: UNIX

Each system message that gets sent to the console is also written to the messages file. Collectively, these messages are an incredibly rich source of information. Savvy system administrators can review the messages file to look for events that might indicate system trouble. Filled file systems, failing devices, and system misconfigurations are just some of the tidbits that can be found in the messages file.

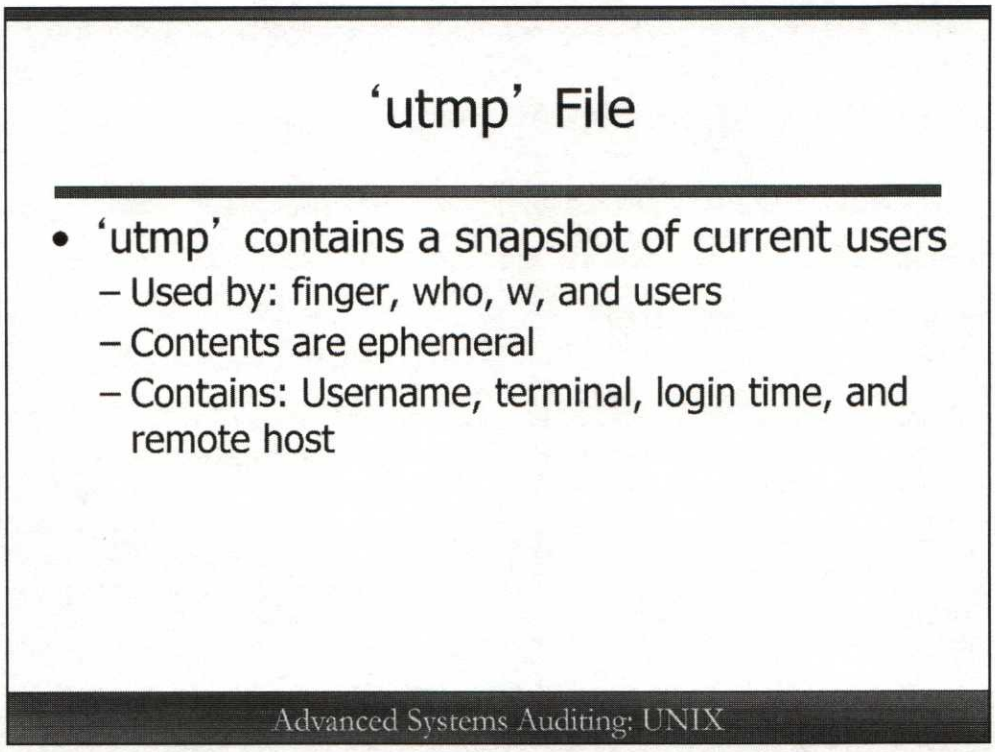
The 'messages' file is more traditionally known as the 'syslog.' Depending on how your system is configured, you might find that the messages file is actually named 'syslog'! It is also possible that your system might have both a 'syslog' and a 'messages' file. In cases like this, it will be necessary to examine the '/etc/syslog.conf' file, which is used to configure what and where things are logged, and to determine what sorts of information you can expect to find in each.

© 2016 David Hoelzer & Enclave Forensics™



The 'secure' log is used to record security and authentication messages on the system. It is quite likely that this information is also recorded in the 'syslog' or 'messages' file, but by creating a second copy in this log, it is easily accessible with minimal effort. The typical applications that will send messages to this file are the 'TCP Wrappers' program, which is used for access control to the system (we will discuss this tool in some depth later), the PAM system (Pluggable Authentication Modules) and the 'login' facilities on the system.

© 2016 David Hoelzer & Enclave Forensics™

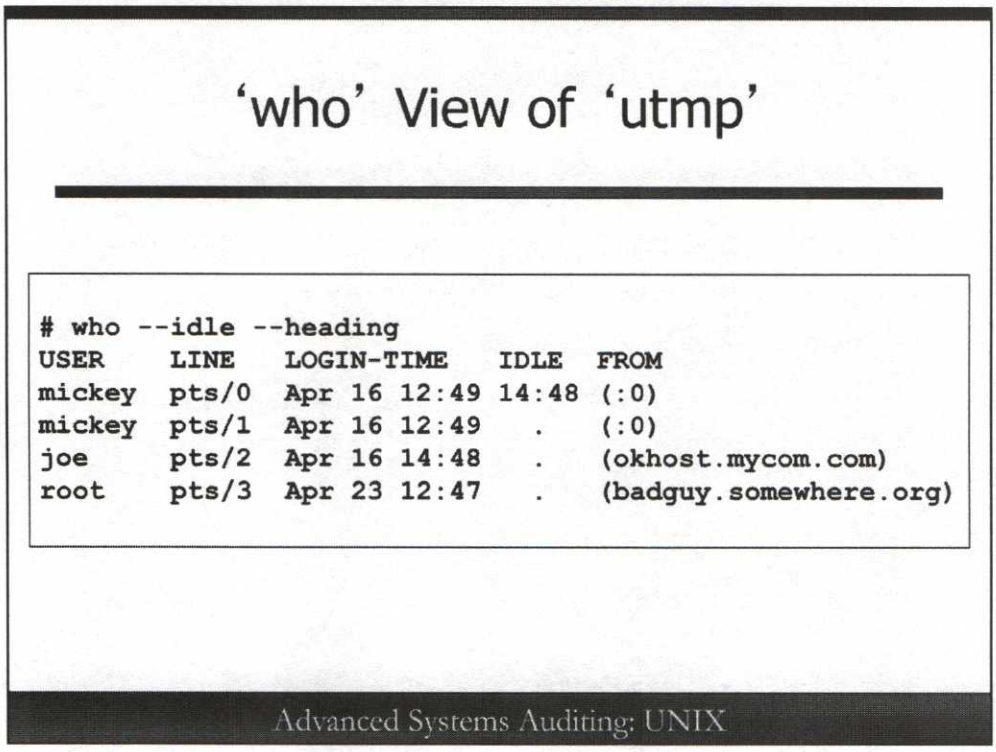


I used to work in a secure facility. Our employee badges had our identity and other pertinent information encoded on a strip on the back of the badge. People authorized to enter a building could do so by swiping their badge through a badge reader. This would record information and unlock the door. The exit process was the same. At any given time, physical security personnel could call up a list of people currently in the building.

The utmp file works in a similar fashion. As users log in, an entry for them is made in the /var/run/utmp file. Upon logout, that entry is removed. The result is a file that contains a current user snapshot at any given point in time. As such, the contents of this file are short lived and always changing.

For every successful login, utmp records, among other things, the username, devicename, time, and origin. Unfortunately, utmp is a binary file, so it cannot be viewed with a text editor or other ascii-based tool. However, programs like who, users, and finger read the utmp file and display its contents.

On the next slide titled, "Examining utmp With who," we look at the contents of an example utmp file.

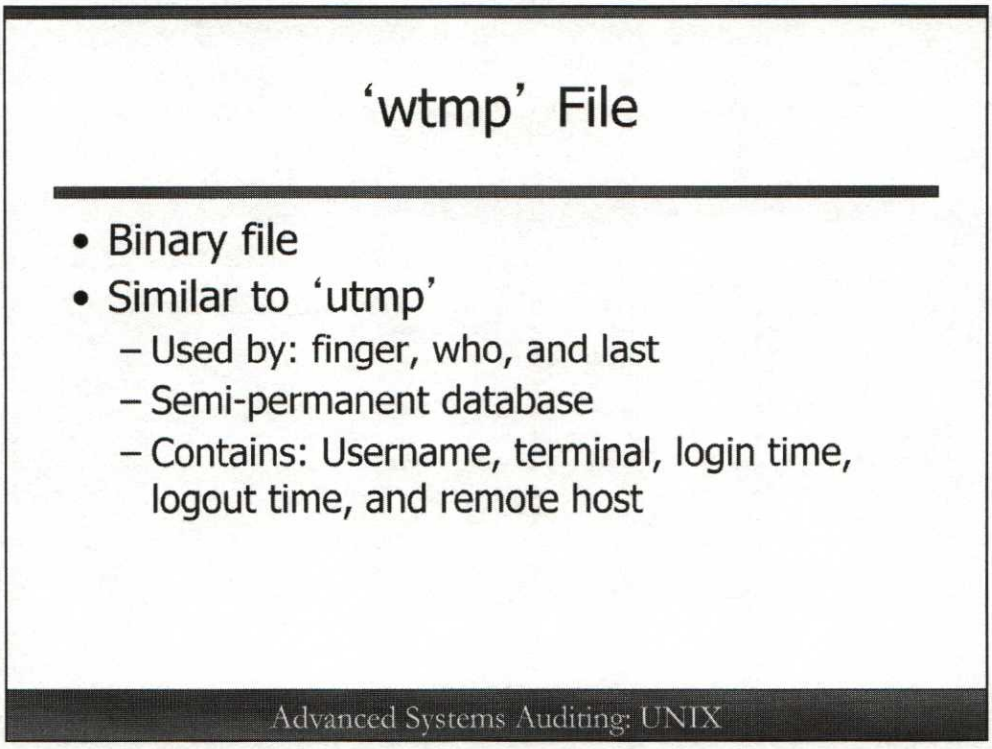


Here, we see the usage of the who command. Several options have been given on the command line, which control the output format, including how long a particular login has been idle.

As advertised, who shows that users mickey, joe, and root are all currently logged in. We can see the terminal lines that were used, as well as the actual login time. Notice that the first entry for mickey indicates that the session has been idle for more than 14 hours, but all of the other sessions are currently active. The final column shows the origin of the login. The :0 indicates that the login occurred on the console itself, so we can assume that mickey sits at this computer. Joe has logged on from okhost.myhost.com, and root is logged in from the remote host badguy.somewhere.org.

It doesn't take a computer scientist to see that something terrible is going on. Someone from an unexpected domain has logged on to our host as root. The initial login occurred at a strange time and the . in the idle column indicates that whoever has assumed root is still actively issuing commands.

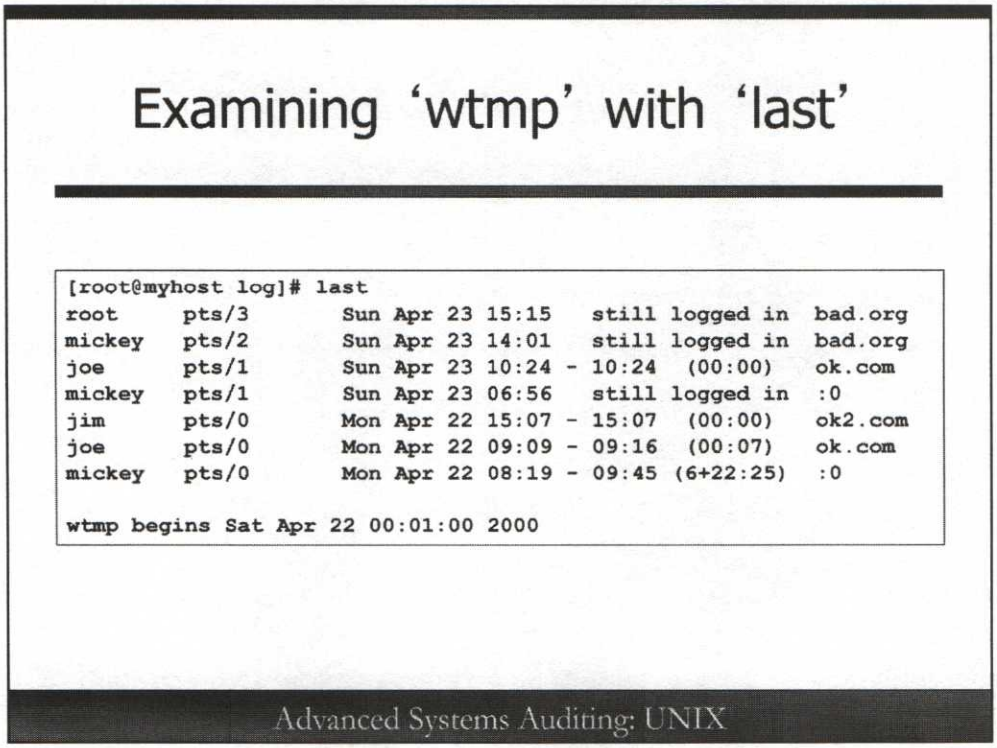
© 2016 David Hoelzer & Enclave Forensics™



wtmp can be found in /var/log, and is the same as utmp in terms of file type and format. It records the username, device, event time, and connection origin as a binary file. The major difference in file content lies in the fact that wtmp keeps a history of all logins, logouts, and system events. This provides a formal audit trail of user account access and host booting. Instead of being a user snapshot, wtmp is a running account and system history. This information is critical to intrusion detection and incident investigation efforts.

The contents of wtmp can be displayed with the who command, but the last command provides much more information. Administrators can prune the results of the who command to display only the last N events; or to show the account events for a particular user.

On the next slide titled, "Examining wtmp With last," we show the last command in action on a wtmp file.



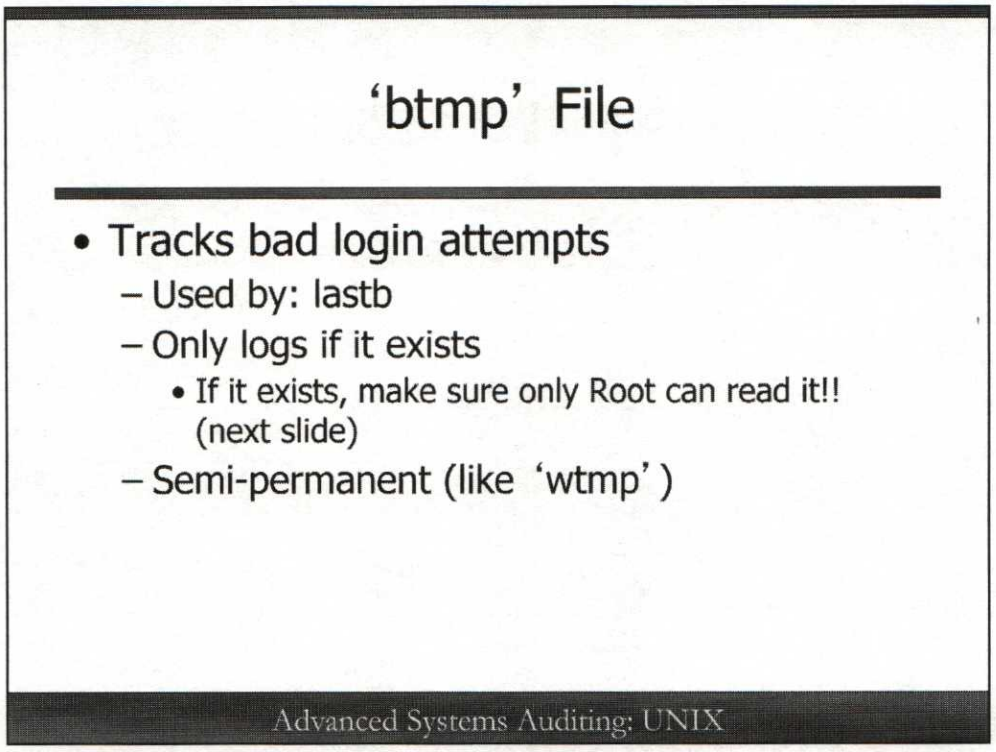
When using last, the first thing to be aware of is that it displays the contents of the wtmp file in a most recent to least recent time order. Said differently, entries at the top of the output occurred most recently. The beginning date of the file is on the very last line. This wtmp began on April 22nd at 1 minute after midnight.

Scanning left to right, we see the username, the tty, the login and logout time, and the host of origin. Elapsed activity time is displayed in parentheses to the right of event times. Also, for those users still logged in, a corresponding message is displayed in place of the logout time.

If we examine the output from the bottom up, we can read a history of the activity on the host. The entries look fairly normal until mickey's entry on April 23 at 14:01. At the top of the output, we see that mickey logged in from the console at 6:56 on April 23rd and was still logged in when the last command was run. The frightening lines are the top two. Apparently, someone is using mickey's account from the host bad.org. In this example, the hacker also acquired a root shell about an hour later. Could a sniffed password have been the vehicle for a local compromise?

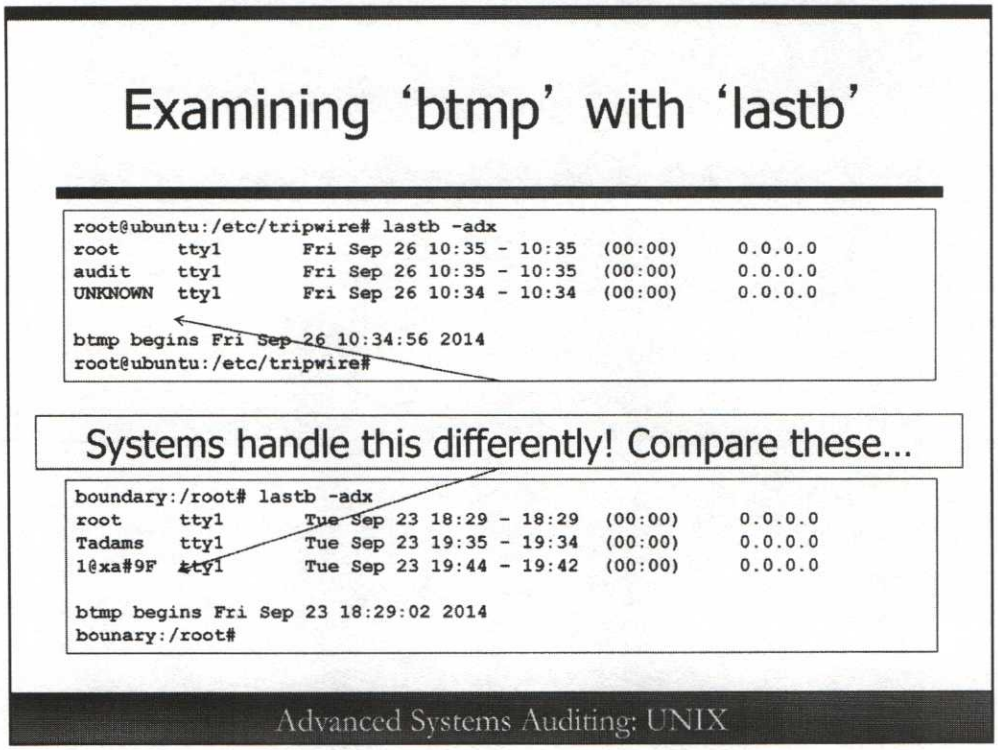
As you can see, the wtmp file is a necessity for reconstructing system events in an investigation. But be warned! wtmp must be activated or no such logs are collected.

© 2016 David Hoelzer & Enclave Forensics™



/var/log/btmp is the file used to log bad login attempts. It records the username, the device, the time, and the origin of the failed login attempt. The lastb command can be used to examine its contents because, like the others, it is a binary file.

Like wtmp, the btmp file must exist in the /var/log/ directory in order for bad logins to be recorded. We can use the ls command to check for the file's existence. If it doesn't exist, create btmp using the touch command as described earlier. Make sure that btmp's permissions are set so that only root can read from it!



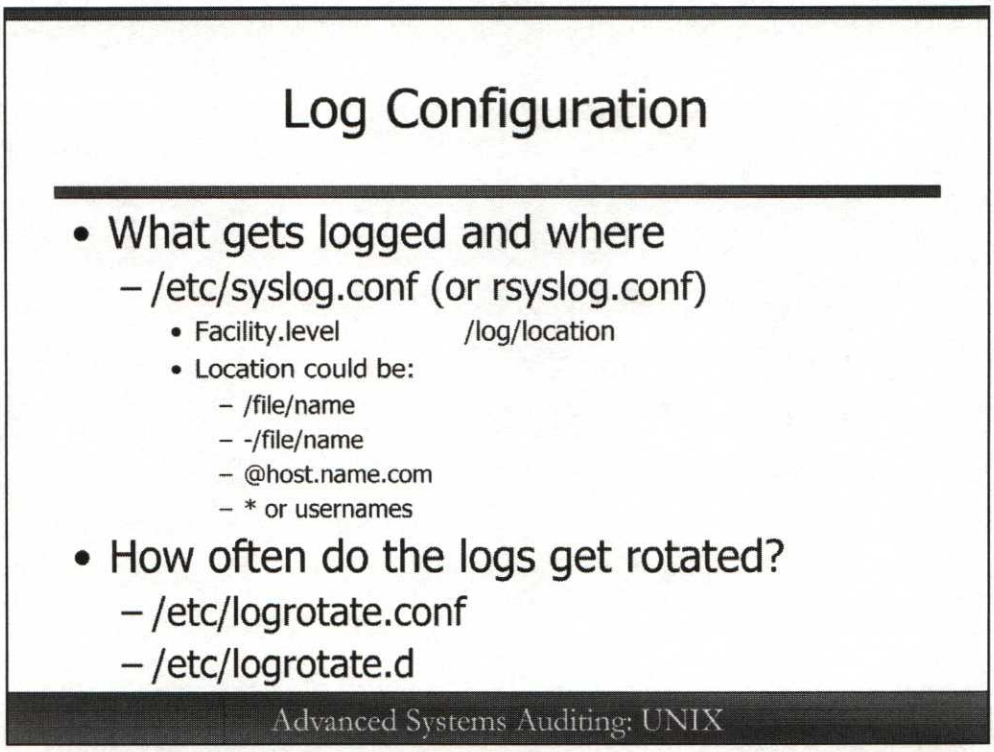
Here, we can see two example outputs from two different UNIX systems. In the first case, we can see the logon failures for a user named "audit," a user named "root" and an UNKNOWN user. Why UNKNOWN? The user entered a value that was not recognized as a UNIX system. This, as it turns out, is a very important protection! Unfortunately, not all UNIX systems have this behavior!

Consider the output in the lower example. Here, we see a system named "Boundary" reporting logon failures for "root," "tadams," and "1@xa#9F". Look at that last username… Does that look sort of suspicious? In fact, it doesn't look anything like a username! What's happening here?

In this case, the btmp is not masking unknown entries. What most likely happened is that a user just accidentally typed his password into the username prompt! Without the masking feature, we end up recording, potentially, passwords!

Remember that we said that only the root user should be able to read this file? This is why. Enterprising users can simply troll this file (if it's readable) for things that look like passwords. When they find one, they can simply look at the output of 'last' to see who logged in right *after* the password hit the btmp file.

© 2016 David Hoelzer & Enclave Forensics™



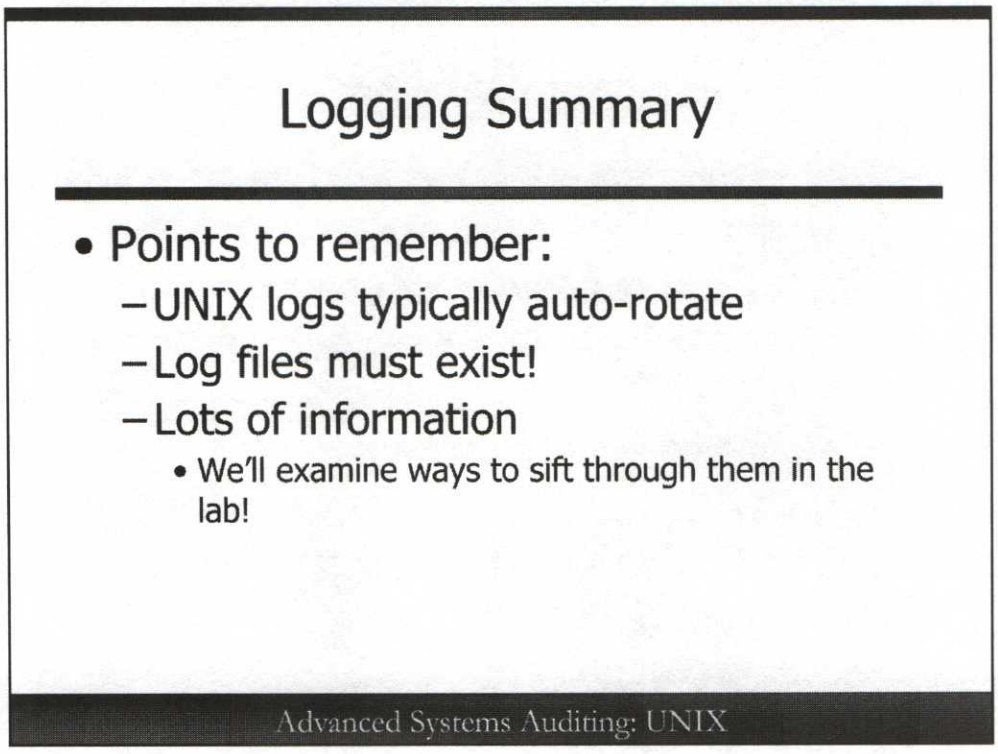
Configuration of the logging is largely controlled through two files. The first is syslog.conf. This is the main logging configuration that defines what gets logged where.

Each configuration line in the file will start with a facility description (kernel, daemon, etc.) followed by a logging level (info, debug, emergency, etc.). These can be lists of facilities and levels or even wildcards. This "selector," if you will, is used to define what will be put into the entry on the right-hand side of the line. To the right, you will typically find a log path and file name. If that file path is prefixed with a '-', the log will not be 'synced' after every write. What this means is that it will cache log entries and write them in batches. For critical events, you would prefer that the log is synced immediately, though this creates more load on the system.

You might also find an @ followed by an IP address or hostname, automatically configuring the system to log remotely. Finally, you could find a username, a list of usernames, or even an asterisk. This will cause the matching log messages to be broadcast to the users listed if they are logged in or, in the case of the asterisk, all logged in users.

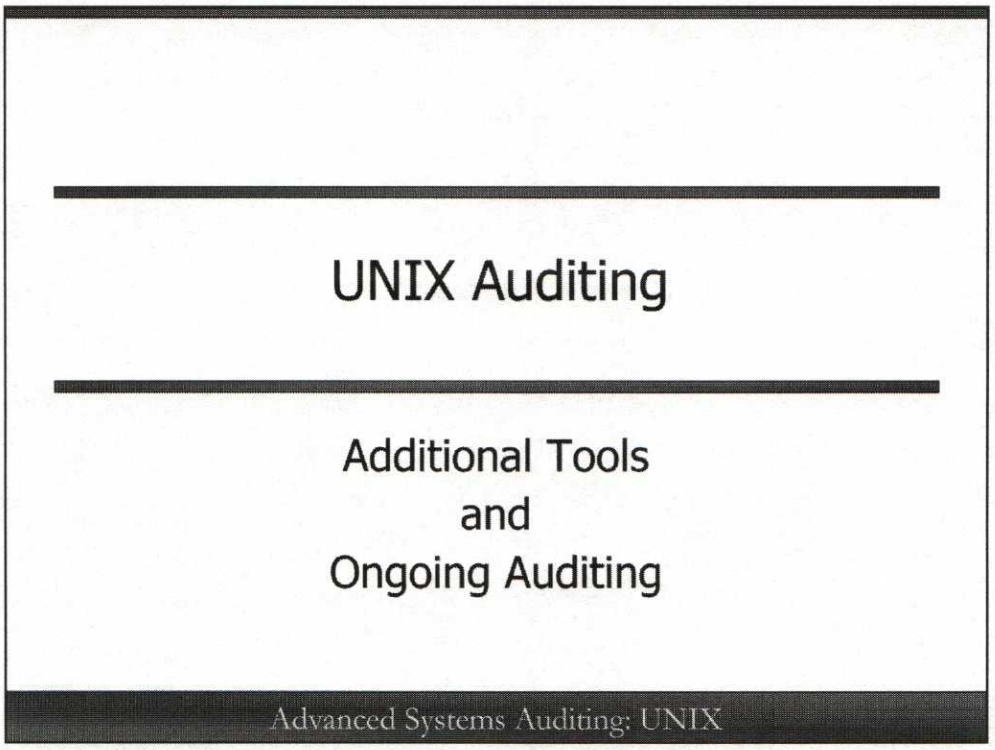
The rotation of logs is controlled by logrotate these days. The primary configuration governing how often to rotate logs and how many copies to keep is found in the logrotate.conf file. Individual configurations for various services can be found in the logrotate.d directory.

© 2016 David Hoelzer & Enclave Forensics™



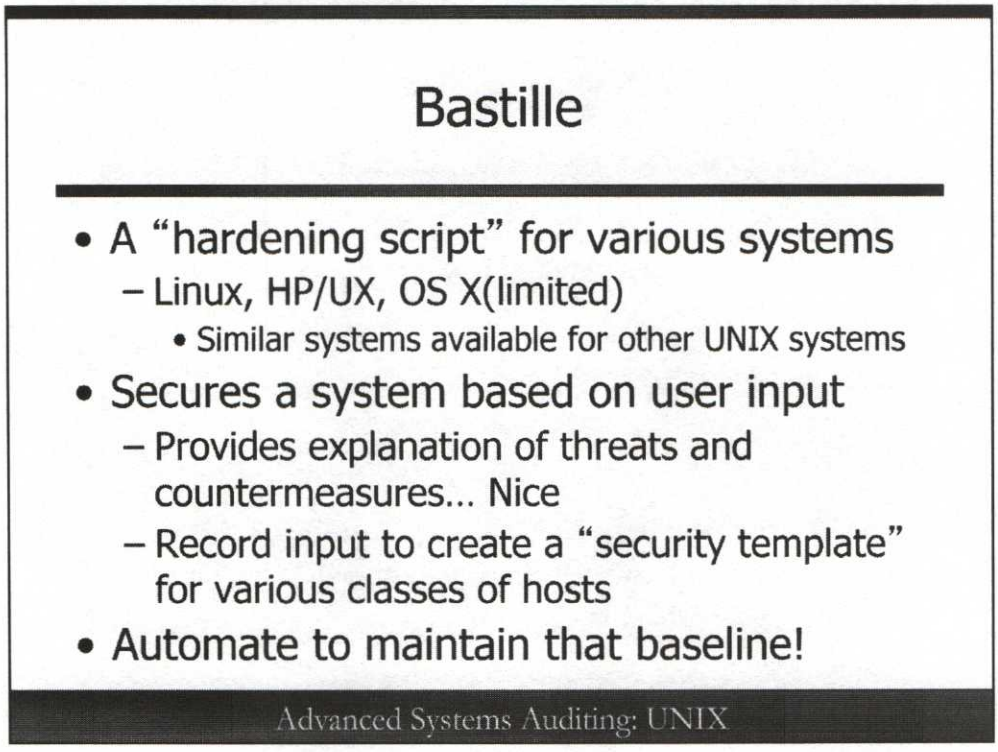
Just a few points to remember so far: UNIX log files are typically configured to automatically rotate periodically, in order for a log entry to be recorded, the log file must exist (UNIX tends to believe that if the log file doesn't exist, you must not want to know), and there's a great deal of information to be found in these logs. We'll examine a variety of ways to trim down the volume of these logs, focusing on the most interesting bits of information.

© 2016 David Hoelzer & Enclave Forensics™



Our next section addresses some of the issues and suggests several tools that can be used to maintain the security of a UNIX system or to audit the system for change.

© 2016 David Hoelzer & Enclave Forensics™

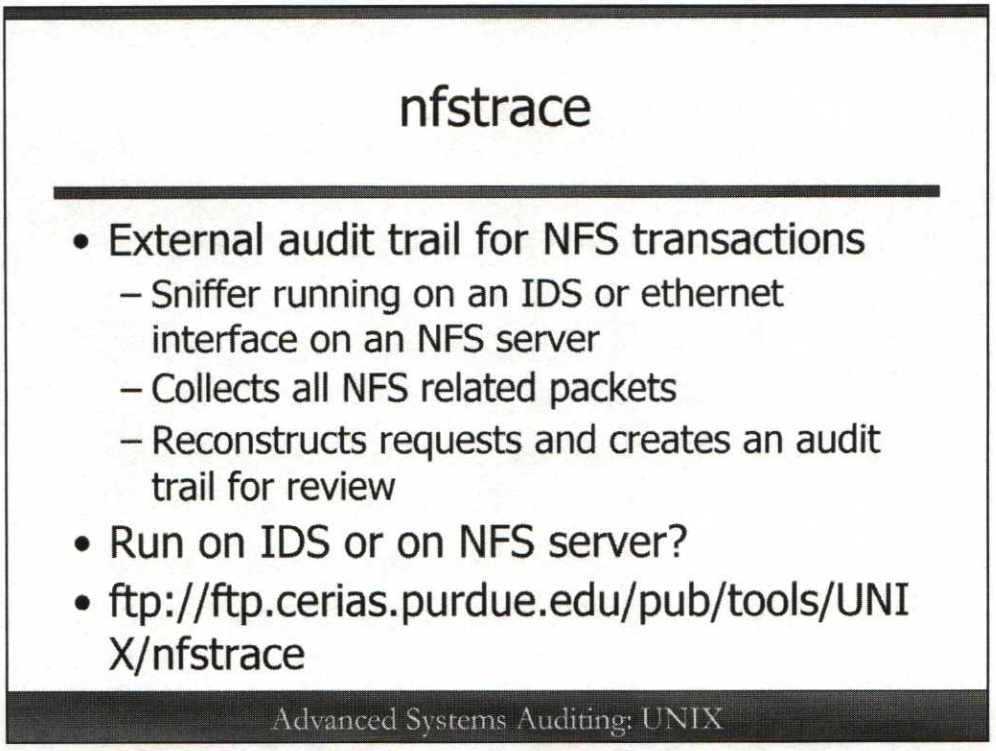


Bastille is available from http://bastille-linux.sourceforge.net and runs on a wide variety of Linux systems. There are also versions available for HP/UX, OS X, and others. Even if Bastille isn't directly supported on a platform that you have, there are other similar scripts available for other operating systems (for example, JASS or SECUR for Solaris). A tool of this type is a must have for these operating systems. Put simply, it's a very user-friendly program designed to harden systems. It provides you with an explanation of the vulnerability or threat, and then asks you whether you want to apply countermeasures (and what those countermeasures might affect).

After you run through the configuration, you have the option to save that security configuration as a template. This template can then be used as input to Bastille from a cron job to ensure that the baseline you define in the template stays implemented over time.

Of course, this is an example. There are many other auditing tools that can be run periodically and report any violations. A partial list is on the next page.

© 2016 David Hoelzer & Enclave Forensics™

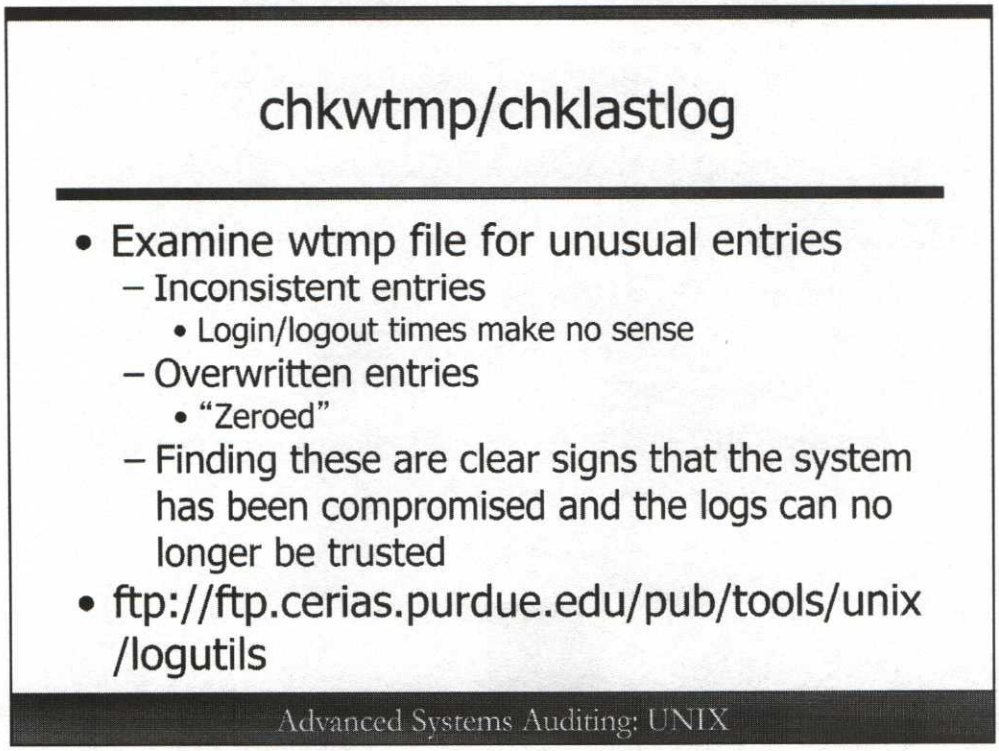


Although UNIX systems in general support the creation of pretty good audit trail information, there are some facilities that don't provide that much detail about the activities on the system. NFS is one such system, but a free tool from Purdue University can be very helpful.

NFS trace allows us to record all NFS-related transactions that occur between our NFS servers and client machines and build a complete audit trail for activities. The software functions by creating a promiscuous listener (sniffer) and collecting all NFS-related packets that pass by. This does mean that you would need to run the software either on the NFS server itself or on a machine that has an interface in the same collision domain as the NFS server. If you already have an existing IDS infrastructure watching your server segment, this would be the perfect place to install this piece of code.

Although it is, perhaps, simpler to get the code running and reporting on the NFS server, we would lose some level of reliability with regard to reporting, especially if the NFS server itself comes under attack or is compromised. On the other hand, deploying the software onto a system to watch the NFS server presents different technical problems. For instance, how are we to collect packets off a switched network?

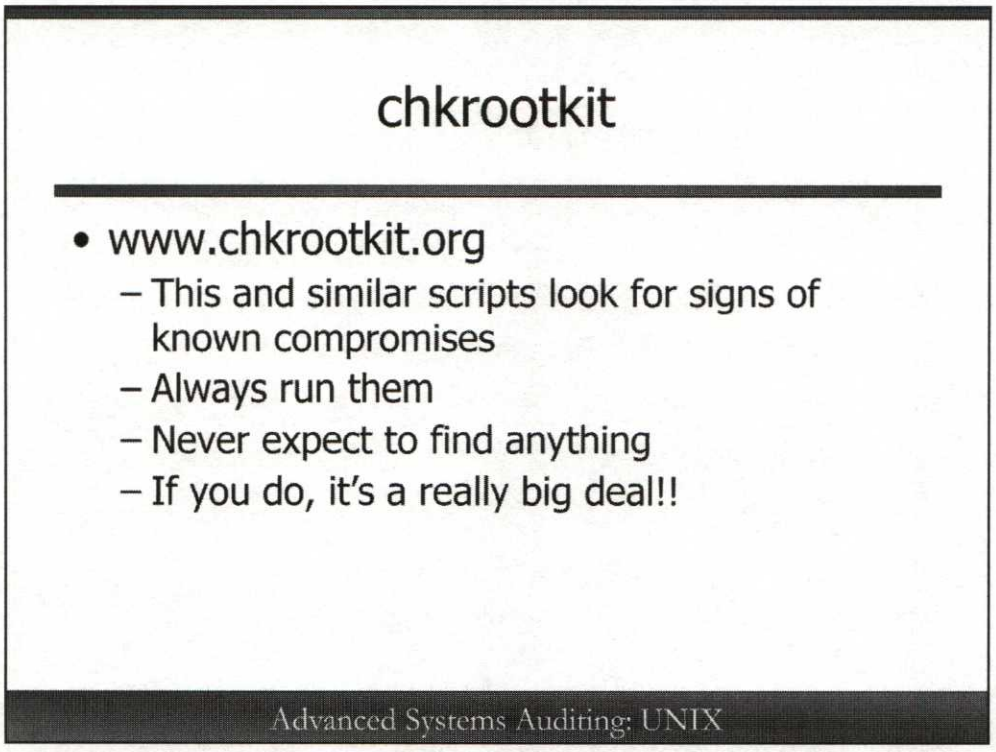
© 2016 David Hoelzer & Enclave Forensics™



Another useful pair of tools is chkwtmp and chklastlog. These two tools can report to you inconsistencies in the /var/log/wtmp file. It reports a variety of inconsistencies, for instance, the logout time occurring before the login time. It also reports to you "zeroed" entries, or instances where the log entry has been overwritten with zeroes in an attempt to conceal a login.

If these tools report anomalies to you, particularly zeroed entries, it is a clear sign that the system being examined has been compromised in some way.

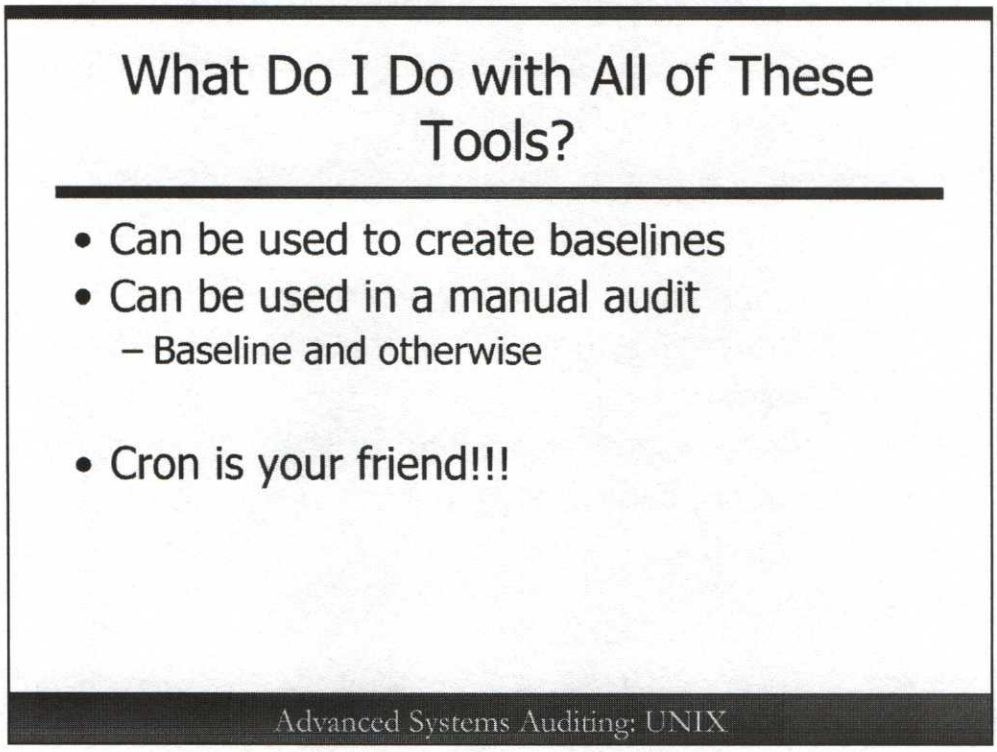
© 2016 David Hoelzer & Enclave Forensics™



Another really handy script that should be run during every audit of a UNIX system is chkrootkit. This is available from chkrootkit.org. This script looks for signs of known rootkits within the file system in addition to performing various tests to see whether everything seems to add up (for example, calculating the actual size of everything on the mounted volumes and comparing that to the amount of space actually in use).

This script is great, but it's really designed to find known vectors. Remember that we're really not in an incident response mode here; we're in an audit mode. This means that you should run this and tools like chklastlog/chkwtmp in addition to things like lsof +L1 during every examination of a system. You should not expect to find anything with these tools, but if you did find something it would either have to have a really good explanation or it would be an extremely big deal!

© 2016 David Hoelzer & Enclave Forensics™

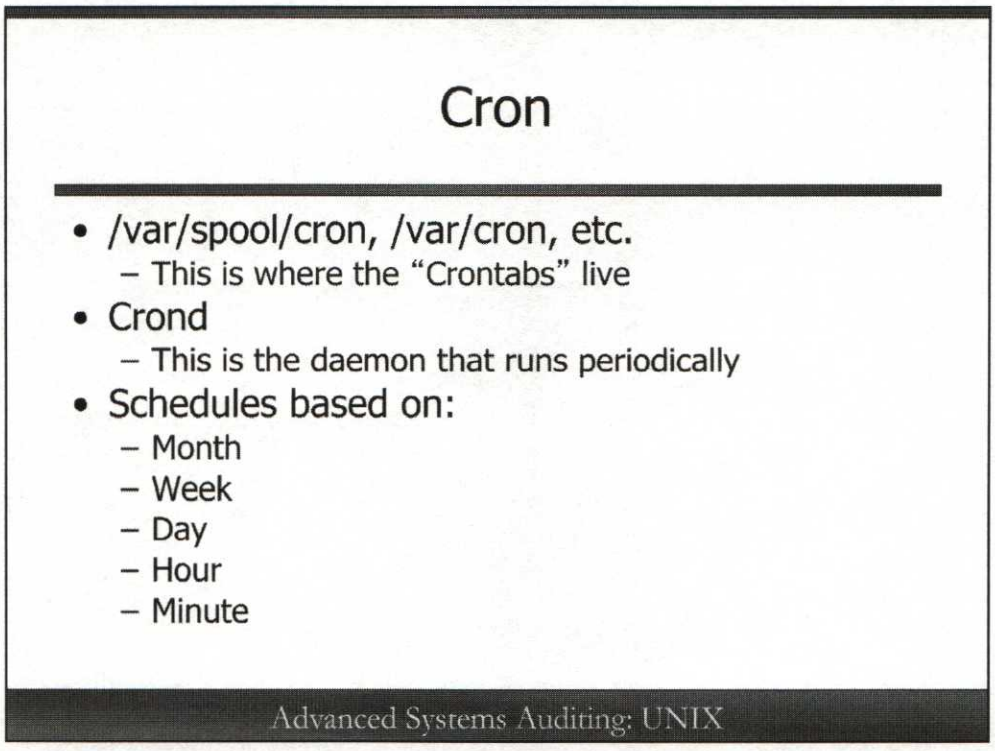


Although all of these tools mentioned on the last few pages can be used to gather baseline information and can be used to perform a manual security audit where baselines don't exist, they are also prime for scripting! If we could script a few of these tools together and have the script run periodically, we could potentially create automated audit reports or exception reports with a minimal amount of effort.

When you begin to create automated reports, it is important to consider the impact that these can have on a system administrator or security officer. If we produce lots of noisy but essentially unimportant reports, the reports will begin to be ignored. On the other hand, if the reports that are created are easy to peruse and report only on exceptions rather than informational "noise," they can become excellent pieces in our overall defense-in-depth or time-based security paradigm.

The UNIX facility that you will want to use to make this happen is 'Cron.'

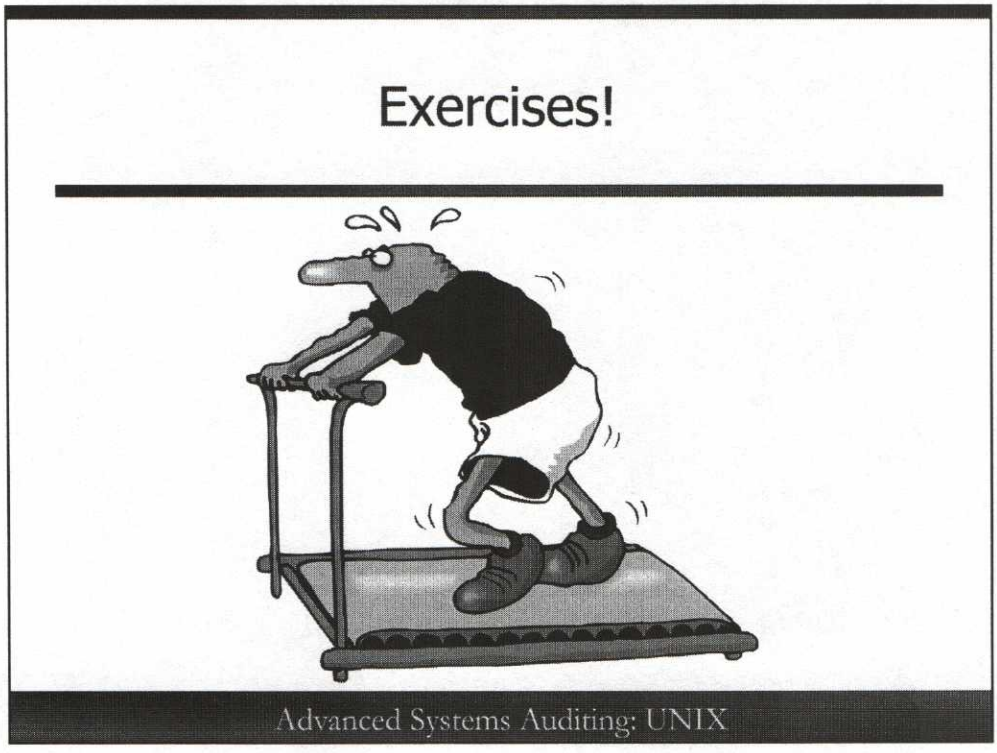
© 2016 David Hoelzer & Enclave Forensics™



The cron daemon is a very flexible scheduling daemon that controls scheduled jobs for all users. Quite likely, even if your administrators don't think they are running cron jobs, there are system cron jobs that run periodically. Usually, UNIX systems will come with automated facilities for rotating logs and performing regular security checks on the system. These jobs are controlled by cron.

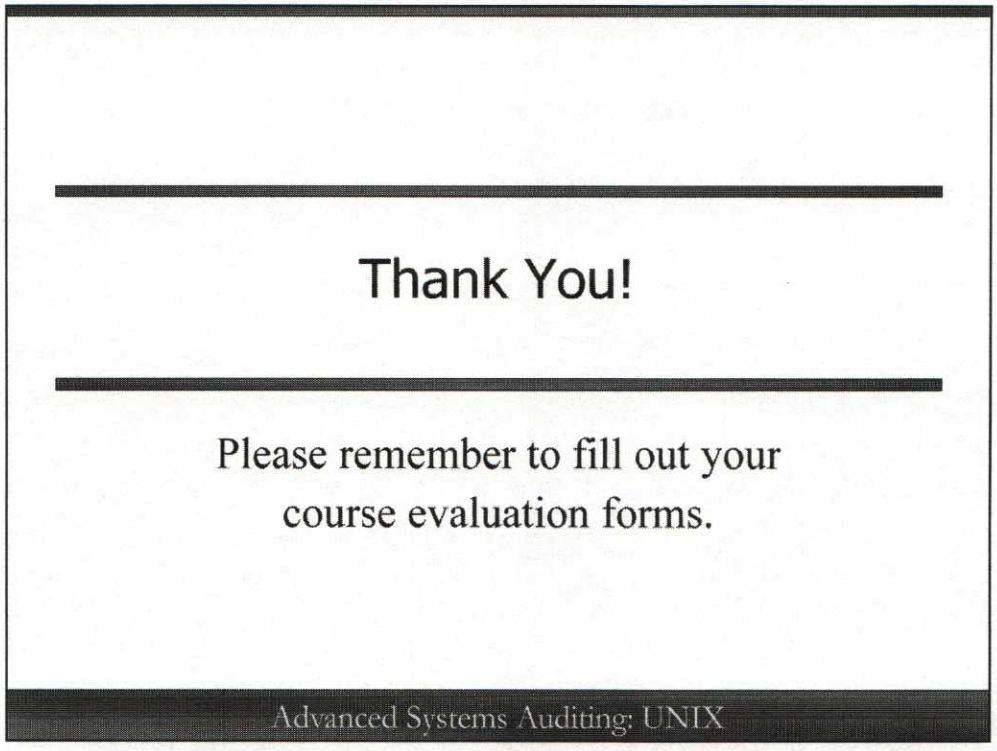
The cron daemon also has a great deal of granularity. Using cron, you can schedule jobs to run tomorrow, next week, next month, twice per month, twice per day, five times per hour, and any other combination that you can think of. This means that we could potentially configure a cron job to run all of our baseline audit tools periodically, say every day or maybe even every hour, and report differences or exceptions to the system administrator or a security officer. The tools that would run are things like Tripwire, chkwtmp, raudit, and so on.

© 2016 David Hoelzer & Enclave Forensics™



At this point, please continue in your exercises book. You should be in a position to complete all of the remaining exercises.

© 2016 David Hoelzer & Enclave Forensics™



This page intentionally left blank.

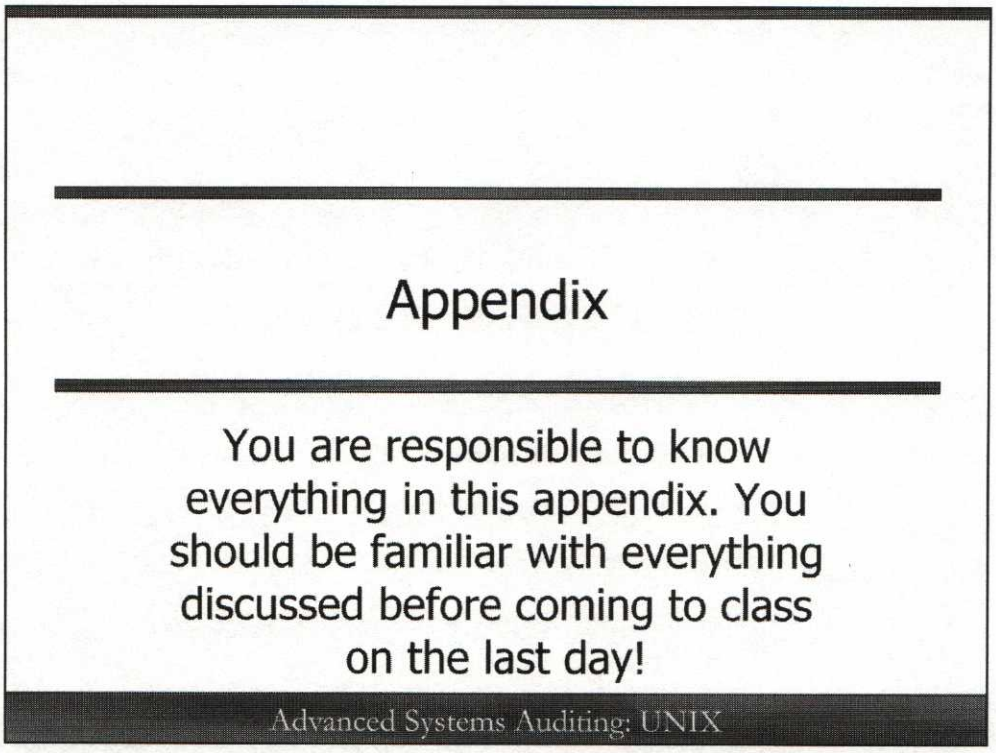
© 2016 David Hoelzer & Enclave Forensics™

Regex Answers

1. Matches entire line
2. "he" or nothing
   - No space in the set!
3. Start of the line through the '2'
   - Matches only one digit followed by zero or more letters
4. Nothing!
   - + requires at least one character *after* a single number
   - Must match zero or more letters at the start of the line
5. Entire line
   - From the beginning of the line...
   - Match anything except 0-9 followed by...
   - One or more digits followed by...
   - Anything except 0-9 to the end of the line

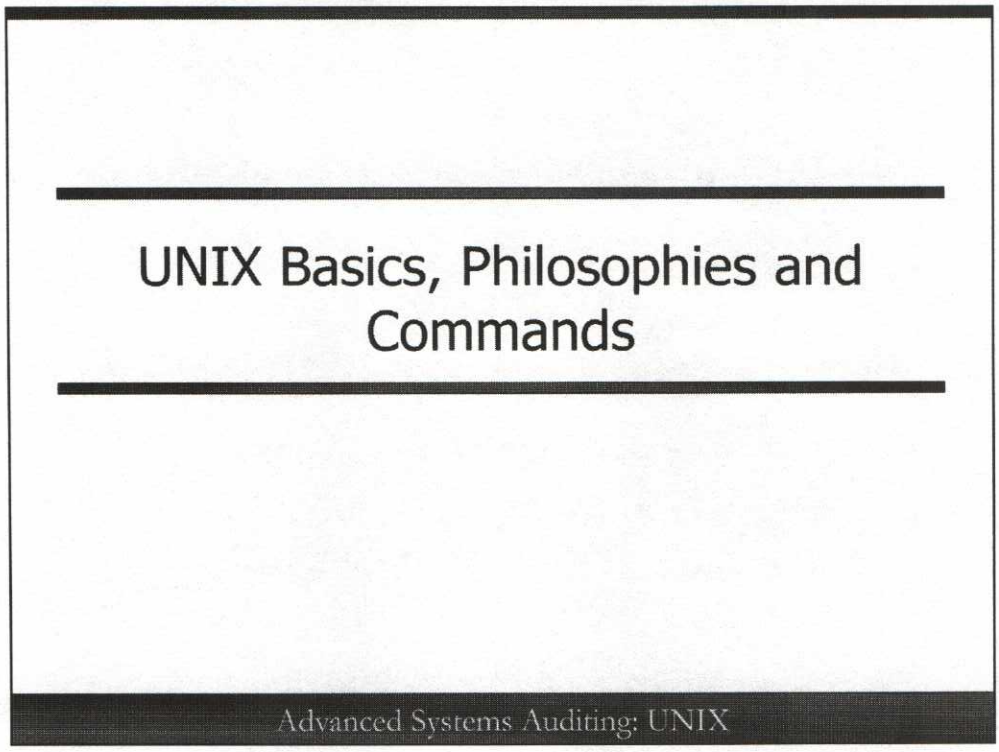
This page intentionally left blank.

© 2016 David Hoelzer & Enclave Forensics™



This page intentionally left blank.

© 2016 David Hoelzer & Enclave Forensics™



This page intentionally left blank.

© 2016 David Hoelzer & Enclave Forensics™

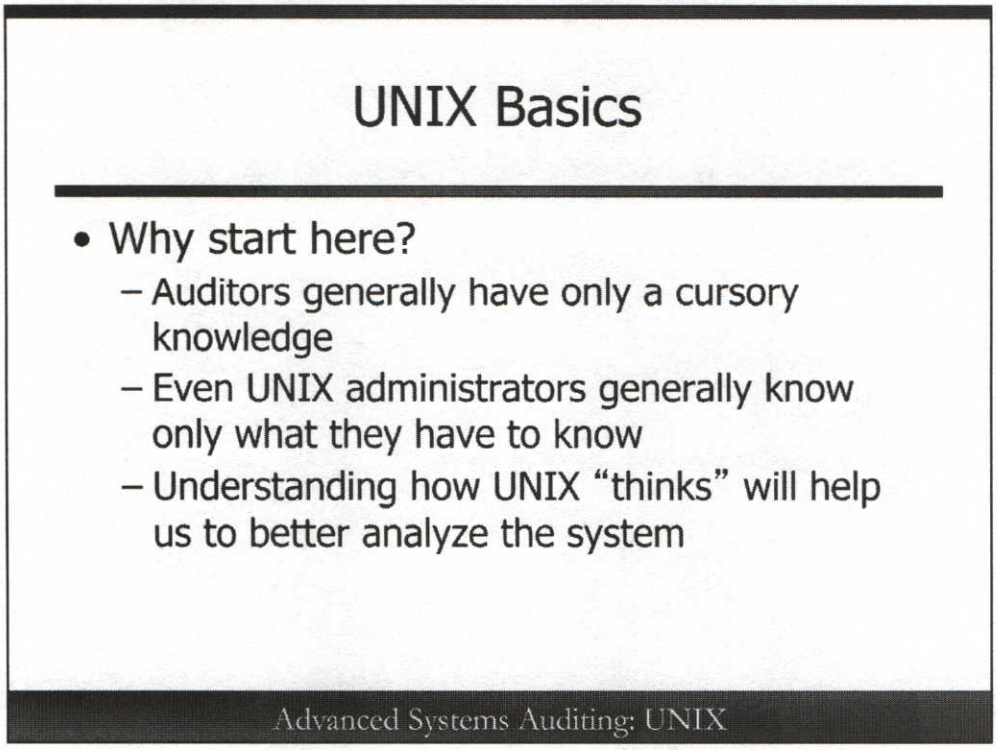
Section Roadmap

- UNIX Basics
  - Types
  - Philosophies
  - Services
    - Portmapper and RPCs
    - NFS
    - X, CDE, and Friends

Advanced Systems Auditing: UNIX

This section starts basic and informational, but will move forward into more and more technical topics in order to give you a firm foundation for understanding why UNIX works the way it does. We cover some very basic history about UNIX and highlight some of the distinctions between the major flavors. We also discuss some of the basic underlying philosophies of UNIX, which will make it easier in the long run to understand what to look for and why. This section also covers a variety of typical UNIX infrastructure components in order to create a framework on which to hang the field work that we describe in later sections.

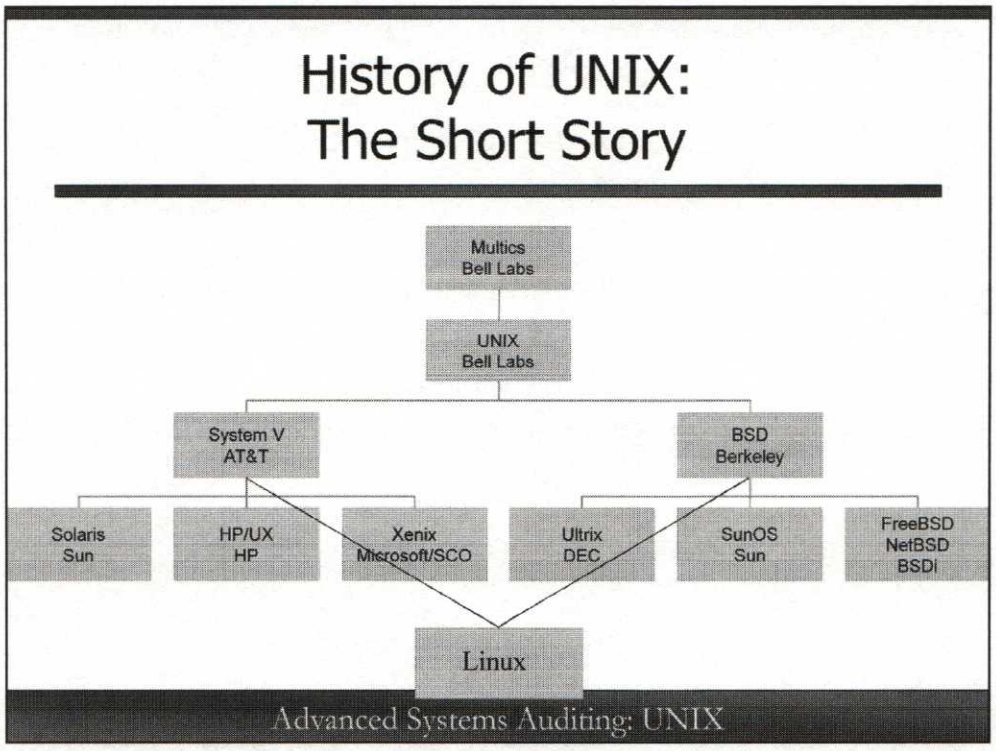
© 2016 David Hoelzer & Enclave Forensics™



You might wonder why, in a SANS intermediate level course, we would choose to cover UNIX system basics rather than diving right into the really deep technical stuff. The simplest answer is that, in our experience, auditors generally have very little hands-on UNIX experience and what they do have is a decade old or more. Even the more technical folks will generally have only very focused knowledge based on the requirements of the environment that they operate in or have operated in. By providing these basics, we can be sure that everyone will be on the same page when we get into the details of analyzing more advanced aspects of the operating system.

Our other primary reason for starting at this level is to help to provide you with a sense of the general philosophies behind UNIX. It would be a simple enough matter to simply tell you to run a series of commands to gather the necessary data from the system that you are auditing, but if we never tell you how the system works, how could you possibly refine or improve the process, much less decide which pieces of information you don't actually need to collect within your own organization.

© 2016 David Hoelzer & Enclave Forensics™



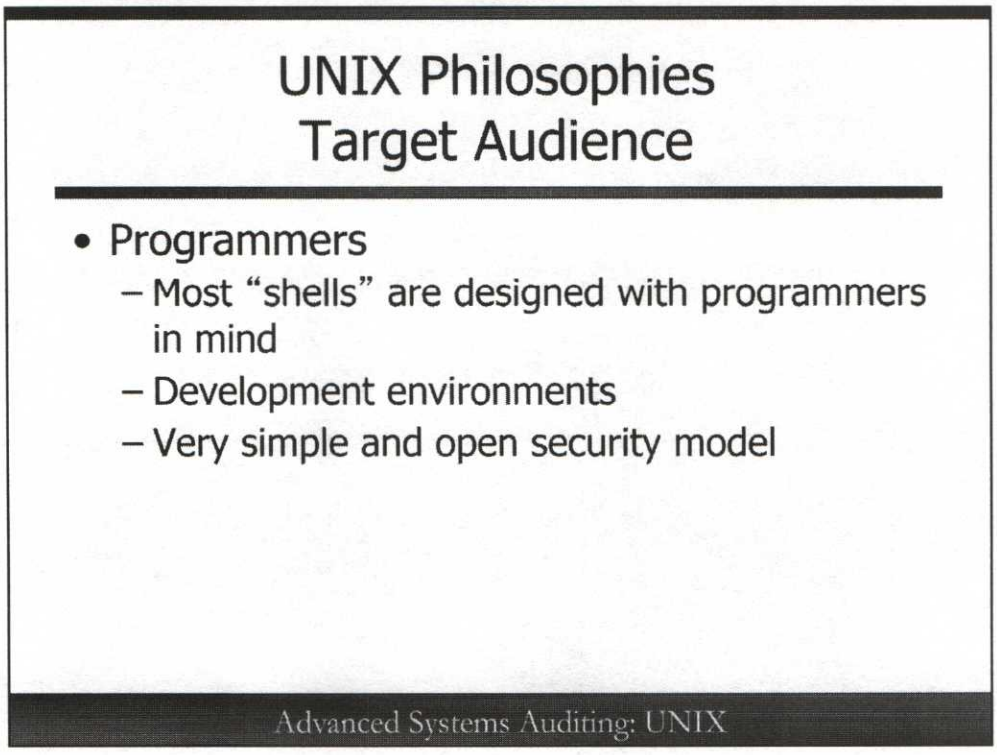
Before giving any other description or explanation, it is important to mention that the chart pictured above is a greatly simplified outline of the general relationships between UNIX distributions. In no way is this intended to be a complete representation of the large variety of UNIX systems that exist, nor is it intended to be 100% accurate. For instance, we acknowledge that a variety of well-known and important operating varieties are not included in our chart, including Irix, Minix, DGUX, Plan 9, and others.

UNIX itself has quite a history, going back thirty years or so. UNIX really had its start as an experimental operating system after funding was cut for the "Multics" project. In fact, the name itself is a bit of a tongue-in-cheek poke at Multics.

UNIX itself continued to be developed at AT&T Bell Labs for some time and eventually two versions of UNIX were born. The version originally written by Bell Labs became "System V" UNIX and the mother of all System V-based UNIX distributions. AT&T, of course, wanted to charge all kinds of money for the source code to UNIX, so some folks at Berkeley decided that they would implement their own version of UNIX. This version became known as the "Berkeley Standard Distribution (BSD)" and was distributed in source code form via magnetic tape to educational and research facilities and just about anyone else who wanted to send them a set of tapes to copy it on to.

The UNIX world continued to evolve for quite some time, but in the early '90s, UNIX began to become more and more available to the masses as a very significant effort to create a completely free version of UNIX was undertaken; Linux. It is true that other free distributions did exist at that time. Minix, for instance, was publicly available, but was difficult to use, very particular about the hardware that it would run on, and not especially efficient.

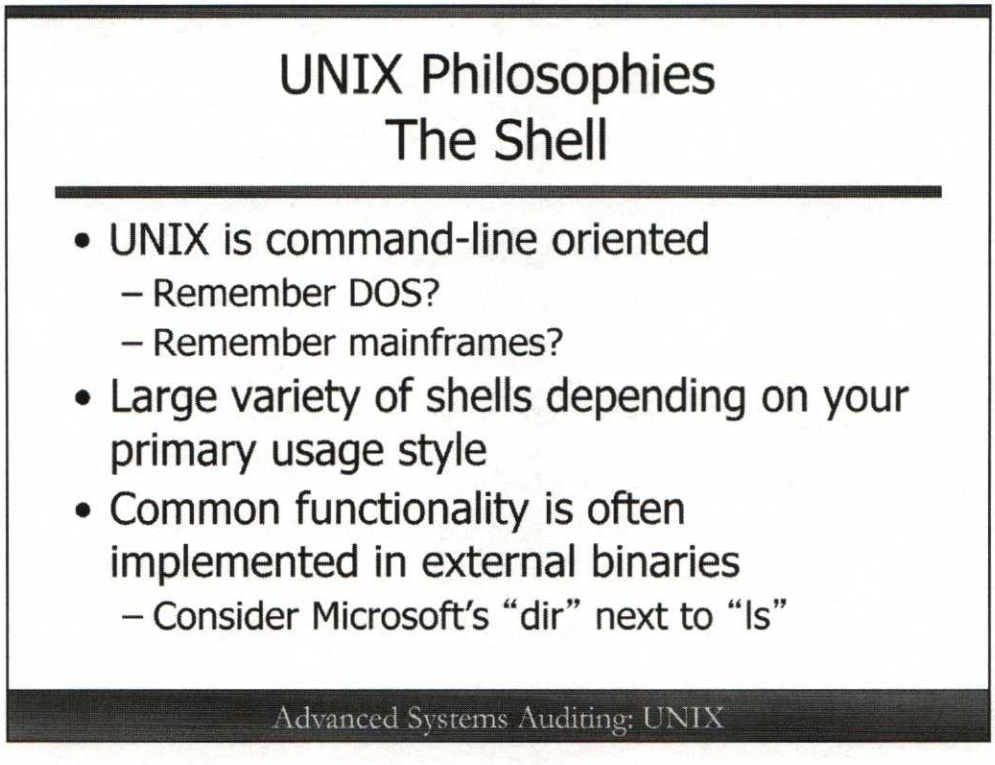
A curious thing about the evolution of Linux is that it has as its parents both BSD and System V. In fact, you can find vestiges of almost every preceding version of UNIX lying around in Linux. The effect of this is that all of the standard UNIX tools are available for Linux, but it's anybody's guess as to whether the individual tools will use the System V style or the BSD style. Essentially, whoever got around to re-implementing the utility or code simply wrote it to suit his needs and to mimic the form of the utility that he was most familiar with.



Regardless of the brand or type of UNIX that you run, UNIX is clearly an operating system designed for real power users or programmers. In fact, almost all of the command shells for UNIX have rather significant scripting or programming capabilities. Not only are these shells powerful scripting engines, but the general use of the shell is clearly designed with someone who knows where he is going and what he wants to do in mind.

Because the target audience for UNIX is programmers and development environments, it is easy to understand why the security UNIX security model is very simple and generally very open. It is possible to obtain add-on security modules for most modern UNIX systems, but out-of-the-box access control is not very granular.

© 2016 David Hoelzer & Enclave Forensics™

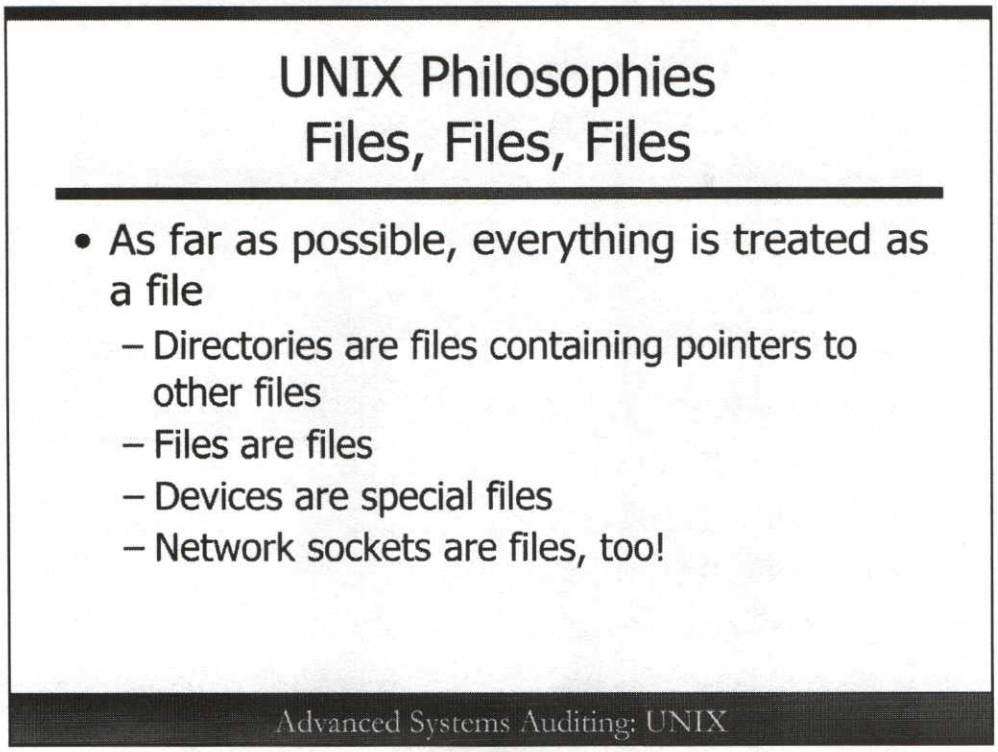


Because we brought up the topic of the UNIX shell, we might as well explain a bit about how the shell fits in with the entire UNIX model. The shell is essentially the user's interface to the operating system. In terms of the history of UNIX, graphical user interfaces (GUIs) are fairly recent developments! Even if you are running or using a GUI, the command line continues to be the primary interface between the user and the operating system itself.

When we consider a GUI under UNIX, it is useful to consider the relationship of Microsoft DOS to Microsoft Windows. Just in case you've forgotten, DOS doesn't mean "Denial of Service." It means "Disk Operating System." When Microsoft first released Microsoft Windows, it was essentially a graphical front end that you could run on top of DOS. Microsoft operating systems have continued to evolve, interweaving the operating system and the user interface more and more closely over time. At this point, the user interface and the command line are so tightly woven together that it is more or less infeasible to use the computer without using the GUI.

UNIX, even today, is essentially still using the Windows over DOS model. Please don't misunderstand, because this is not necessarily a bad thing! Consider, for instance, the role of a firewall. How often does someone actually need to sit at the firewall console and do interactive work? If we were to run that firewall on a Microsoft platform, we would have no choice but to use and run the GUI, which consumes considerable resources. On the other hand, the same firewall running under UNIX does not need to run the GUI, thereby allowing for a more efficient use of system resources overall.

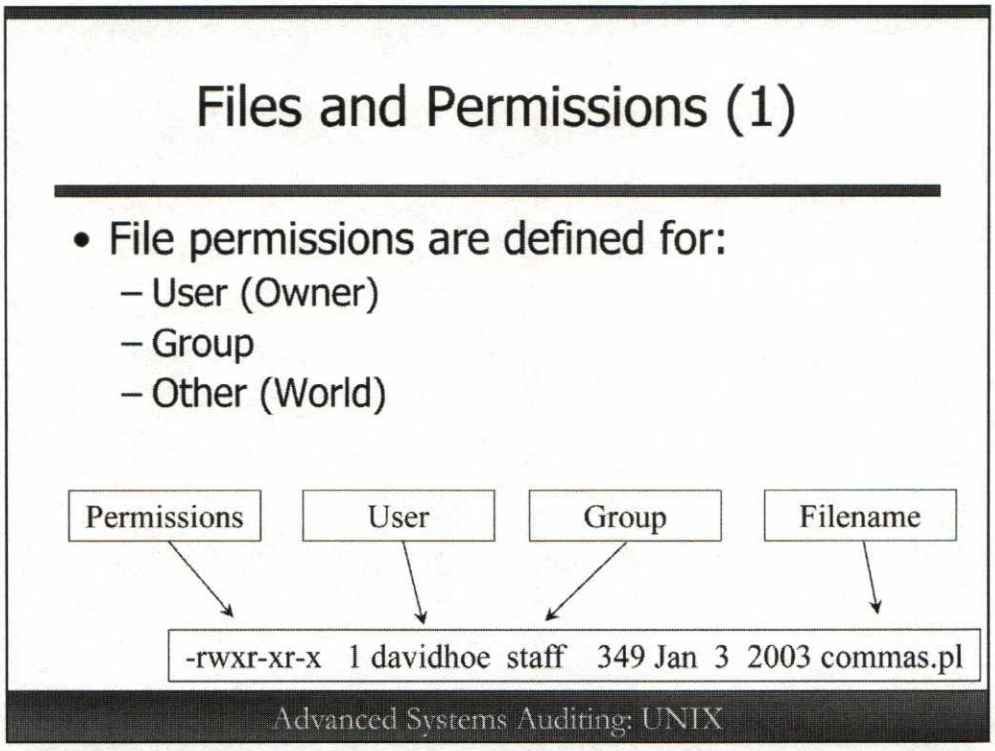
In terms of the UNIX shell, there is another interesting fact to be aware of. Although the UNIX shells have a great deal of functionality built into them, common functionality is typically implemented externally. The best way to explain this is to consider the example of the "DIR" utility under Microsoft. To obtain a file listing, you execute the DIR command. This command is actually a part of the cmd.exe program on the Windows-based computer, which means that if you want to obtain a listing from a program, for instance, you essentially need to either start a command shell and execute DIR there or implement your own directory routine. Under UNIX, however, shell programmers realized it was a waste of effort to duplicate code again and again, so the 'ls' command is a separate binary that is executed as a sub-process. This means that any program or process on the system can call the 'ls' utility at any time, eliminating the need for you to re-implement the command in your own code…unless you really want to.



A pretty useful thing to know about UNIX is that, as far as is possible, it treats everything as a file. For instance, a directory itself is simply a file that contains a list of filenames and "inodes," index nodes, and MAC times (Modification, Access, and Creation). Even hardware devices that are connected to the system are, in essence, treated as files. Typically, you will find links to these devices in either the /dev or the /devices directory, depending on the operating system. These "file" entries can be used to directly manipulate the device. Recent years have brought the /proc file system. This special file system even allows you to treat processes, process memory, and some other interesting aspects of processes as files!

Although this might sound like a strange arrangement, it will actually work to our advantage. The UNIX paradigm makes it much simpler to determine which resources are being used by an individual process because there is a single interface to interrogate rather than many possible methods for accessing a resource.

© 2016 David Hoelzer & Enclave Forensics™

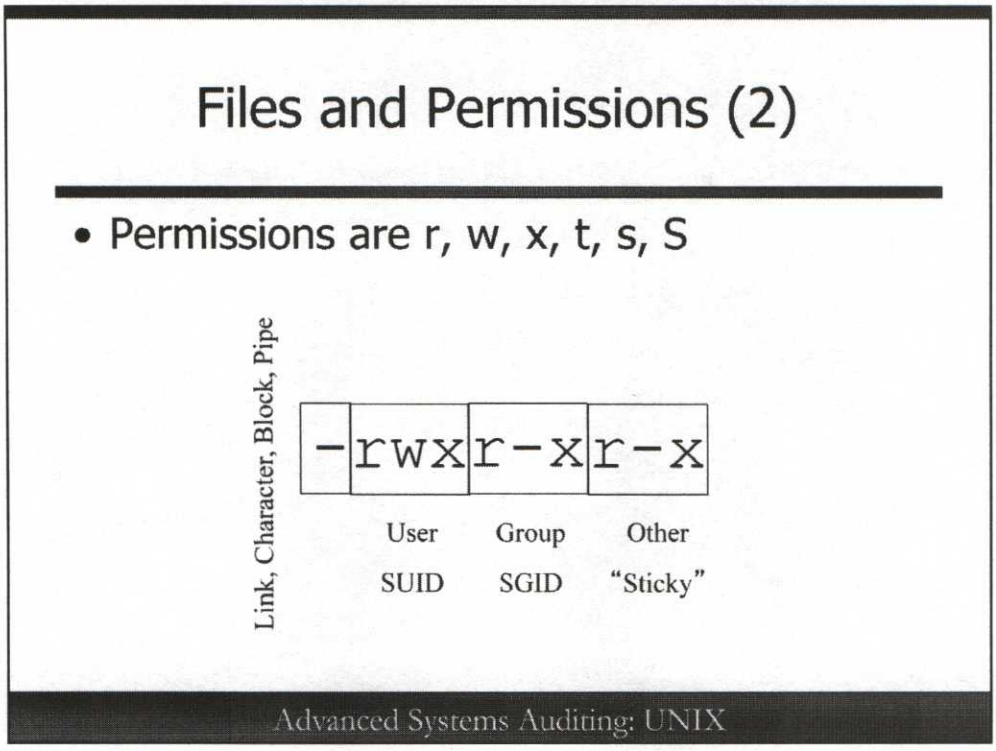


Because we're already on the topic of files, it seems that we should mention something about access controls or file permissions. Although many modern operating systems have very complex and granular mechanisms for granting or restricting access to files and other resources, UNIX uses a very simple system. In terms of permissions, it is only possible to specify access controls in terms of what the user (owner) can do, what members of the group that owns the file can do, or what the world (or "others") can do.

In its simplest form, file permissions amount to Read, Write, and Execute permissions. There can be some interesting side effects to these permissions that we'll get to shortly, but in essence we are able to define the access the owner, group, or world have to a file based on those simple terms. There are a few other special options that can be used with file permissions as well.

It is possible to set the permissions on a file so that the person running the system inherits the effective user id (EUID) of the user who owns the file. This is known as a set-uid (SUID) program. Essentially, this means that the process thread under which the program runs has all of the rights and permissions of the owner of the file. The /bin/passwd program, for example, is typically a SUID program owned by root; this is more or less required because only the root user may alter the password file on the system.

© 2016 David Hoelzer & Enclave Forensics™

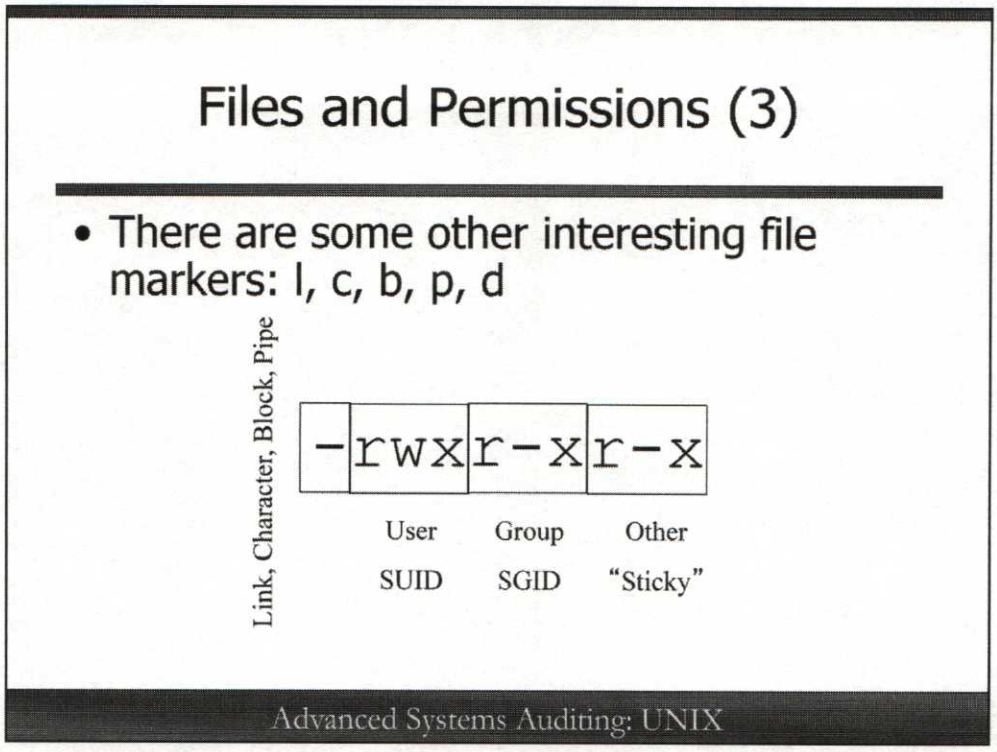


Because there is only so much room for representing the file permissions, some of the positions do double duty. Logically, because a SUID program changes the EID, the 'x' for execute in the Owner section is replaced with an 's' for SUID. Similarly, if this were SGID, the Group 'x' would be replaced with an 'S.' The world 'x' is used to represent whether or not the 'sticky' bit is set. The sticky bit is mostly historical, though there are some modern uses. Unfortunately, interpretation of the sticky bit is operating system specific, so we will not attempt to describe all of the possible functions of this bit.

The most common usage of this bit today is for directories. Imagine that you have a server, perhaps a web server, where multiple users are involved in editing and uploading files to the shared web directory. An extremely common problem is that one of the users replaces a file in that directory and the default UMASK (user permissions mask) that is applied sets the permissions in such a way that none of the other authorized individuals can modify the file. At the same time, the file is set to be owned by the default group that the user is a member of, so the group permissions are also too restrictive.

To resolve this problem, one of the users reaches out to the administrator to have the group ownership of the file changed. Although this is a short-term fix, it is really not the correct solution. The best solution is to set the sticky bit on the *directory* that the file resides in. The sticky bit used in this way will enforce the group ownership of all of the files in this directory. What this means is that when a new file is created by a user, that file will have the folder's group ownership regardless of what the default user's group is!

© 2016 David Hoelzer & Enclave Forensics™



The first character listed with the file permissions is not actually used to define the permissions but the actual type of file. The five possible values for this position are l, c, b, p, or d. Each of these values is used to indicate that this is a "special" file.

Intuitively, the letter 'd' in this position indicates that this file is a directory. The other values are equally intuitive in that they represent the first letter of the term that they replace:

l = Link

c = Character device

b = Block device

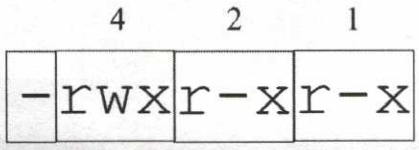
p = Named pipe

The only one of these types that is of special interest to us is a link or symbolic link. Links are used to create pointers to files within the file system or even to files on other physical file systems. There are actually two types of links: hard links and soft links. A discussion of the actual differences between these two is beyond the scope of our discussion, but the purpose of links is to allow you to create an easy reference point to a file that is physically located elsewhere. Effectively, this allows the file to appear to exist in more than one place at a time without consuming additional resources.

© 2016 David Hoelzer & Enclave Forensics™

Files and Permissions (4)

- Permissions are natively octal
  - Octal = base 8 (0 - 7)
  - Read = 4
  - Write = 2
  - Execute = 1
- SUID and SGID bits take a "macro" view to apply the same strategy
  - SUID = 4
  - GID = 2
  - (Sticky = 1)



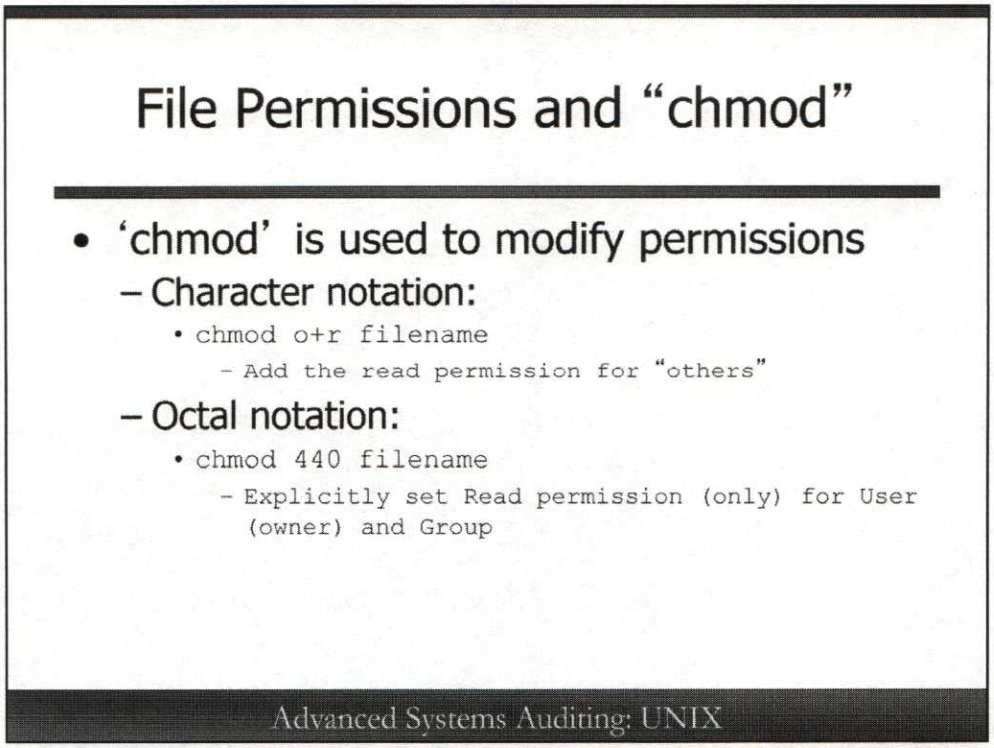
Advanced Systems Auditing: UNIX

Within the file system itself, the file permissions are not actually stored as letters, but as base 8 or octal digits. This allows us to use any of three possible bit values to represent each of the read, write, or execute permissions. This means that if a file had both read and write permissions turned on but execute permission was turned off, the value for that permission would be 4+2 = 6 (Read = 4, Write = 2). This also means that we can have an octal value from zero through seven in any of the three positions that define the permissions for the owner, the group, or the world.

In reality, there is a fourth octal value available at the most significant or left-most position. This octal value is used to control the special SUID, SGID, and Sticky bits. In a somewhat abstract yet logical way, the values of the SUID, which makes us appear to be the owner, is 4, the SGID bit is valued at 2 and the sticky bit is valued at 1. In other words, if we look at the overall permissions structure as three possible bit values (owner, group, world), we simply assign the proper bit positional value and arrive at the method used internally to represent these bits.

I know that sounds really technical, but we mention it now because much later today, we will need to understand how these bits work to use an extremely important tool: find.

© 2016 David Hoelzer & Enclave Forensics™

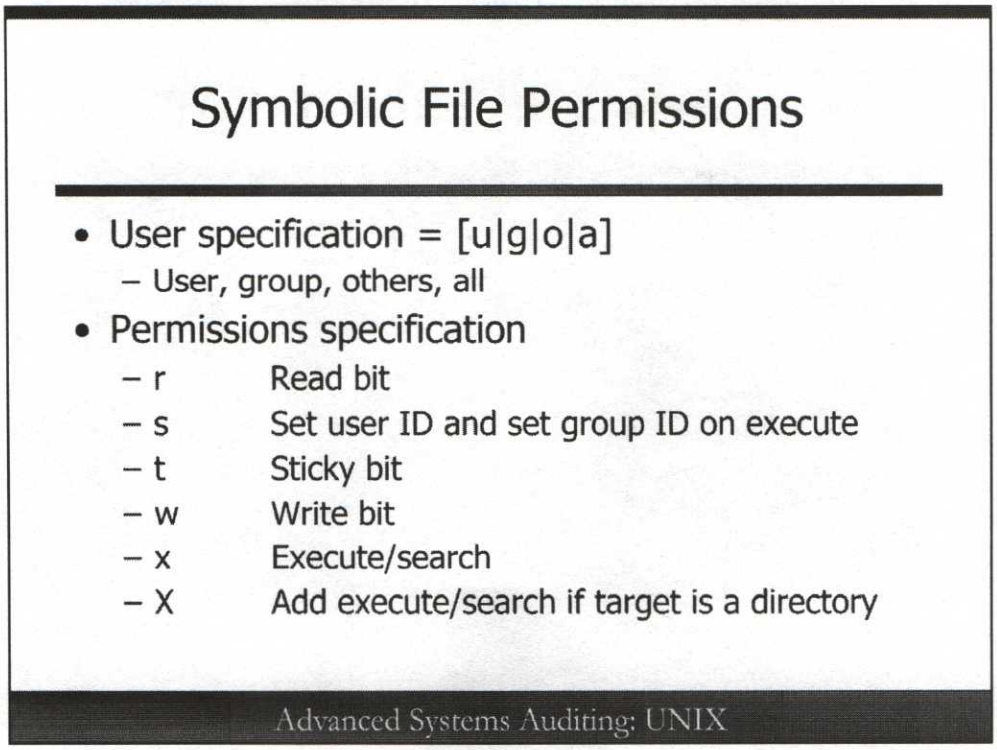


The command-line mechanism for adjusting the permissions that apply to a file or directory is the 'chmod' utility. When using this utility, we have the ability to specify the permissions to set or change on the file in terms of a simple character notation or using an octal representation (remember that octal stuff from the last slide? :)).

Using the character representation, it is a simple matter to add or remove specific permissions. The octal representation requires that we specify exactly what we would like the permissions to be. Here's the general format used for setting symbolic file permissions:

chmod [u|g|o|a[u|g|o[u|g|o]]][+|-|=][r|s|t|w|x|X|u|g|o] [filespec]

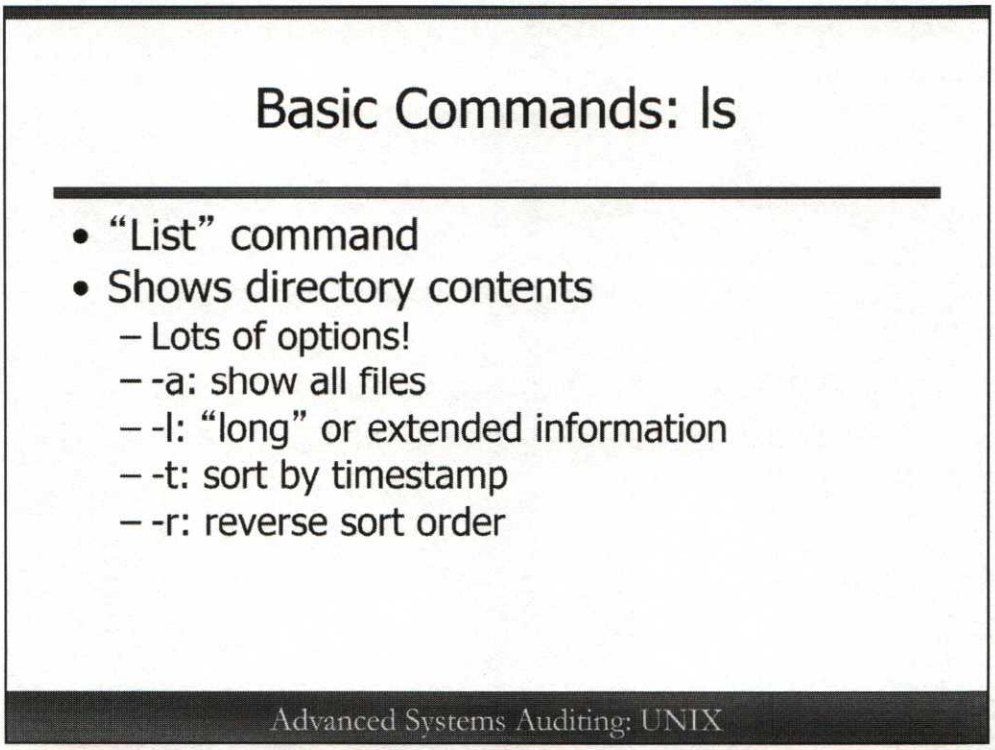
© 2016 David Hoelzer & Enclave Forensics™



The symbolic permissions are more flexible than the numeric permissions in that it is easier to modify a single bit or selective set of bits across a selection of files without changing the permissions to an explicit set of permissions. When making changes, we can specify those changes to apply to the owner (u), the group (g), others (o), or any combination of these. To change all three, we can use the 'a' symbol to represent that we're changing all of them. 'a' is a synonym for 'ugo.'

The actual permissions are pretty easy to understand. We can choose to add (+), remove (-), or set explicitly (=) any of the typical read (r), write (w), and execute(x) permissions. There are also symbols to allow us to set the sticky bit, which is appropriately represented by a 't' because that is how it is represented in the file permissions display from 'ls.' We can also turn on the set user and set group ID bits (s), though there are no consistent settings that allow us to target just the set group or just the set user ID bit. Finally, there's a pretty handy 'X' permission that will set the execute permission only if the target is a directory.

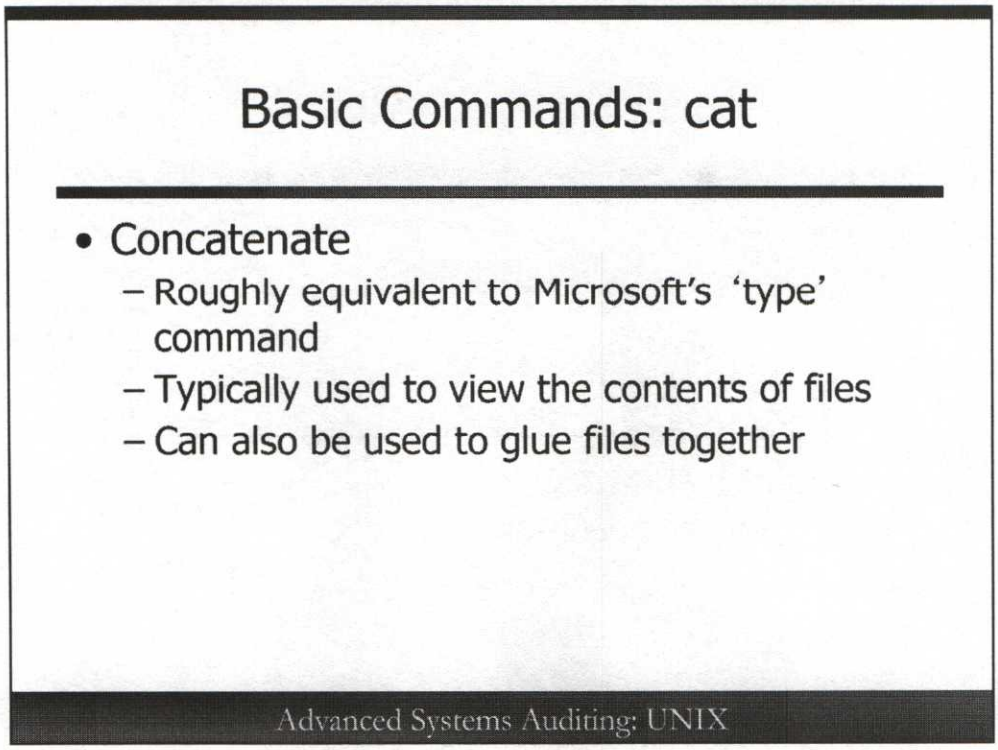
© 2016 David Hoelzer & Enclave Forensics™



The 'ls' command is probably the command that you will run more frequently than any other command at a UNIX command line. Quite simply, 'ls' stands for "list." 'ls' allows you to list all of the files that are in the current directory or potentially in the directory that you specify. Like most UNIX tools, 'ls' has lots and lots of possible options. Even if you've been using UNIX for some time, please take some time during the exercise section to actually leaf through the manual page for 'ls'; chances are that even seasoned administrators will find some handy feature that they were not previously aware of or completely forgot about.

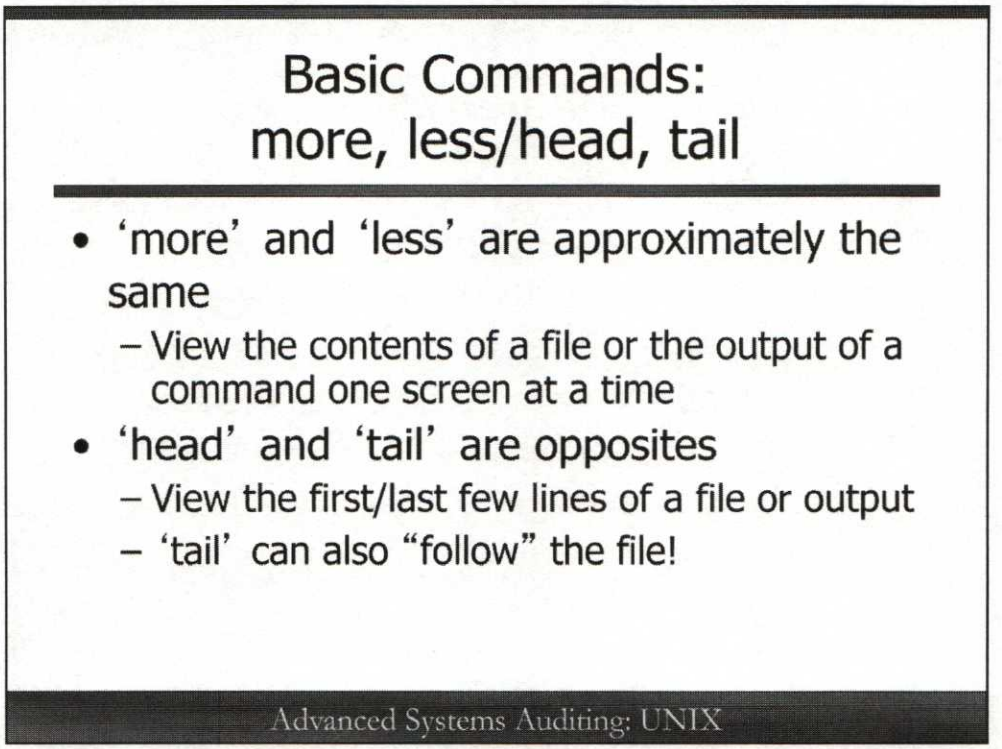
Some very useful and common options, especially for auditors or incident handlers, are the '-a' to show all files (including hidden files—a file is considered "hidden" if the name begins with a '.'), the '-l' option (to obtain a "long" listing that includes file modification time, permissions, owner, group, link count, etc.), the '-t' option (which lists files according to the modification date), and the '-r' option (which is used to reverse the sort order of the files). The normal sort order for a directory listing is alphabetical.

© 2016 David Hoelzer & Enclave Forensics™



The 'cat' command is an abbreviation for "concatenate" and is roughly equivalent to the Microsoft 'type' command. The 'cat' utility can be used to quickly view the contents of a file, send the contents of a file through some other utility as input, or glue files together, which is the actual definition of concatenate. This tool is useful on its own as long as the contents of the file are less than one screen in length or if what we want to see is at the end of the file. If the file is longer, there are a few other tools we should know about.

© 2016 David Hoelzer & Enclave Forensics™

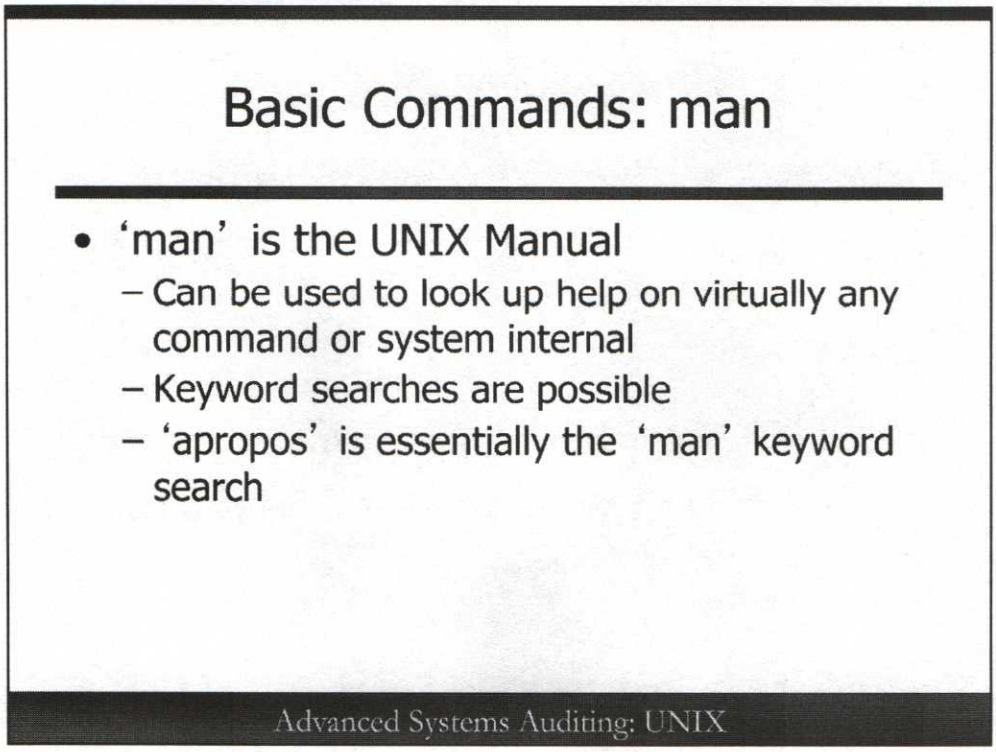


Not all files will be small enough to view using 'cat.' To solve this problem, we have a variety of nifty tools to handle these situations.

The 'more' and 'less' tools are more or less the same. They are both used to view text files or command output one page at a time. If you are using these tools and would like to advance to the next page, the space bar will skip ahead. You can also press the Enter or Return key to move ahead one line. In many cases, you might also want to use the 'b' key to go back one page. When you are finished viewing the file or output and want to leave 'more' or 'less,' just press the 'q' key to quit.

Our other two tools, 'head' and 'tail,' are opposites of one another. These utilities can be used to see the head of a file, typically the first five to ten lines, or the tail of a file, the last five to ten lines. You can, of course, specify exactly how many lines you would like to see; you can even specify the line offset that you would like either tool to start displaying from, relative to the head or tail of the file. 'tail' has one additional feature that is incredibly useful when it comes to monitoring logs. It is possible to tell 'tail' to "follow" a file, which means that it will display the last few lines of the file in question and then begin to wait for more text to be written into the file. As new lines are appended to the file, 'tail' will display them for us. The '-f' option is used to put 'tail' into "follow" mode.

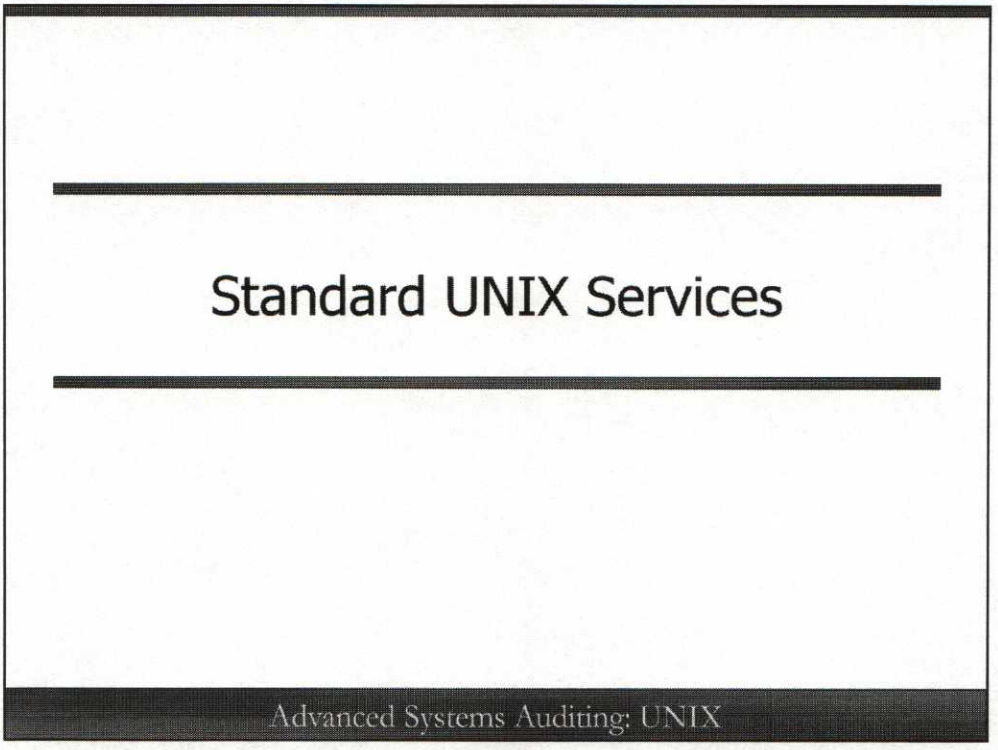
© 2016 David Hoelzer & Enclave Forensics™



If you are new to UNIX or if it has been a long time since you actually used UNIX, your head might be spinning already! It turns out that you really don't have to remember all of this stuff. Very early on, the creators and developers of UNIX decided that it would become extremely cumbersome to create a printed manual. In fact, you can buy UNIX manuals for commercial Unices, but both the free and commercial distributions have maintained electronic manuals as well. These manuals, accessible using the 'man' command, are accessible from the command line in UNIX. There are also a number of GUI-based manual page viewers like 'Xman.'

The 'man' command might not be useful if you had to know the name of the command that you wanted help on first. To this end, you can also perform keyword searches using either the 'man -k' command or the 'apropos' command. Either of these will allow you to enter keywords to search for, and the manual system will report any potentially useful manual pages that are installed on the system.

© 2016 David Hoelzer & Enclave Forensics™



For the remainder of this section, we examine some of the most common UNIX services and programs at a very high level. This information will prove to be useful later when we begin to audit the system for common flaws and unnecessary services.

© 2016 David Hoelzer & Enclave Forensics™

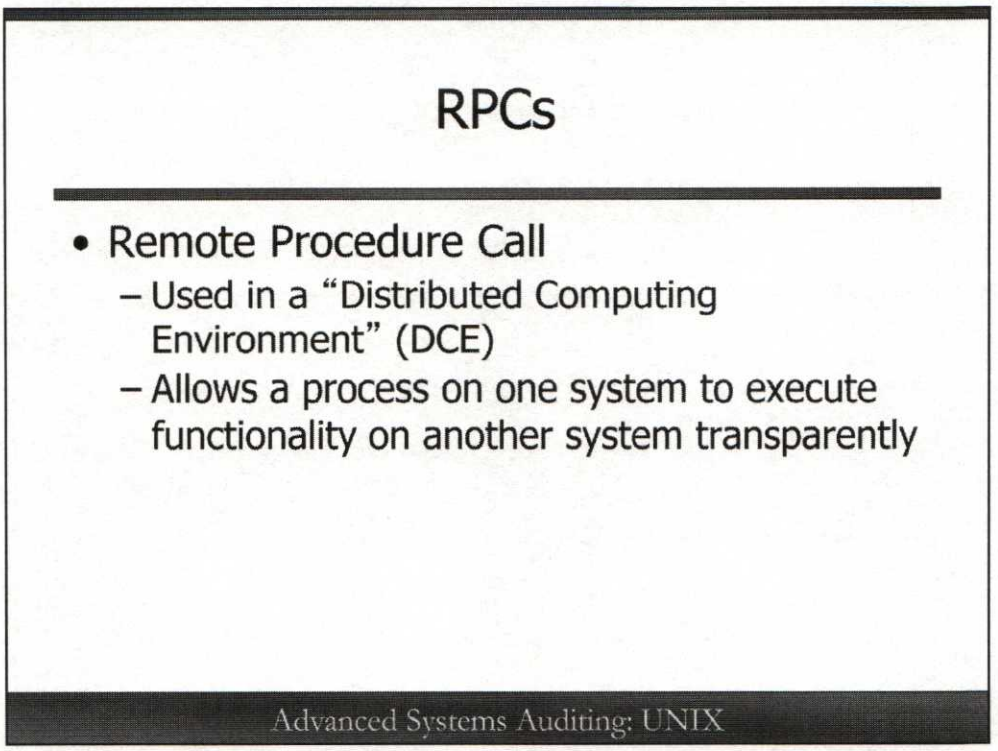
UNIX Services

- Standard Services:
  - Portmapper and RPCs
  - NFS
  - X, CDE, and Friends

Advanced Systems Auditing: UNIX

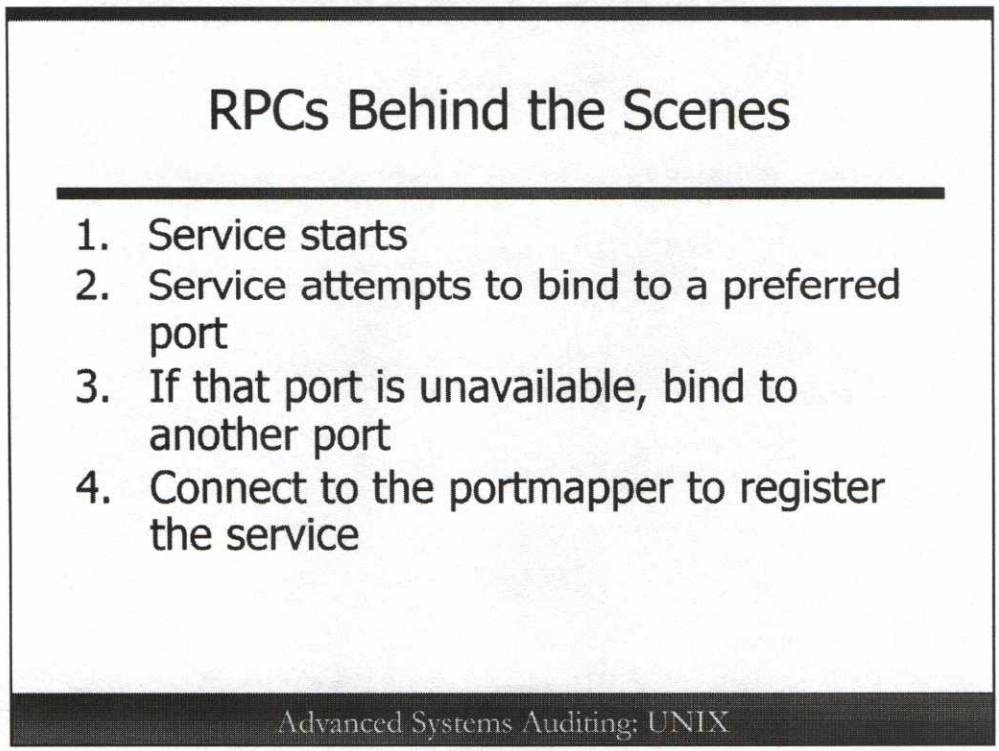
Although there are many, many possible services, RPCs, NFS, and X-Window are likely the most commonly found. We touch on a variety of other services throughout the day and point out a variety of standard security configurations for a variety of other services (like web servers, mail servers, etc.).

© 2016 David Hoelzer & Enclave Forensics™



Remote Procedure Calls (RPCs) are extremely common in a UNIX environment. These services are intended to allow you to implement a centralized software resource or service on your network that all other systems in your network can access. They can also be used to create a distributed computing environment, allowing you to have another system perform back-end work while the system that the user is sitting at handles the GUI or other tasks. RPCs are used to implement a large variety of services including NFS and aspects of the Common Desktop Environment (CDE).

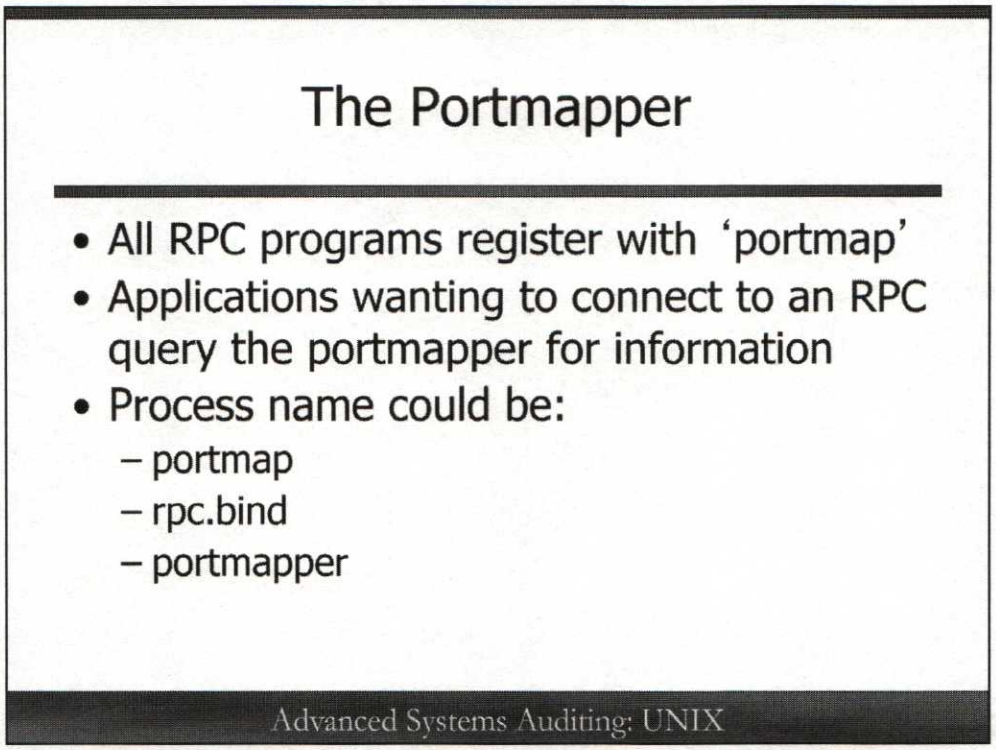
© 2016 David Hoelzer & Enclave Forensics™



Behind the scenes, when an RPC service starts, it begins by attempting to bind to its preferred port. For instance, when the NFS service starts, it will attempt to bind to port 2049 by default. It is possible that port 2049 is not available, however. If the port is not available, the RPC service will either attempt to bind to another preferred port or it might ask the network stack to allocate any port and report back to the RPC service. Whichever strategy is taken, you can imagine that this could lead to problems; how can we find the NFS service if it's not running on the standard port number?

This problem is solved by the final step. When the service successfully binds to a port, it then registers with the portmapper.

© 2016 David Hoelzer & Enclave Forensics™

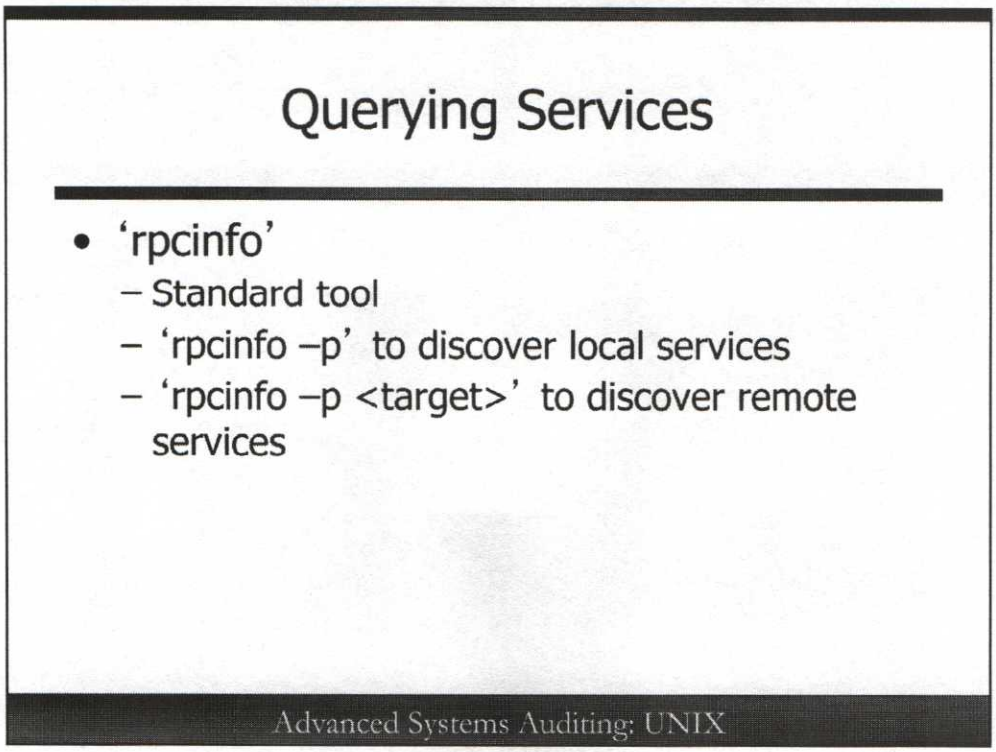


The portmapper itself is also an RPC program, but it always listens on port 111 TCP and UDP. The portmapper on certain operating systems might also listen on some other high-numbered ports, but this is not especially important for our discussion at the moment.

The portmapper essentially acts as a directory service, allowing applications to register their versions and the port numbers that they are listening on. Programs that need to access one of those services can query the portmapper to find out whether the service is running and what port numbers it is available on.

The portmapper process itself might have a variety of names depending on the system that it's running on. Common names are 'portmap,' 'rpc.bind,' and 'portmapper,' though you might find others.

© 2016 David Hoelzer & Enclave Forensics™

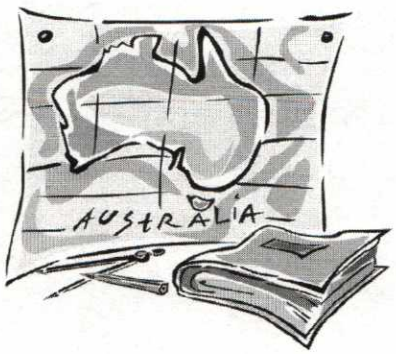


It is possible to test which RPC services are available on a system by using the 'rpcinfo' tool. This tool connects to the portmapper and performs an RPC "Dump," This means that the portmapper is asked to return a list of all available RPC programs, the names of the services, the protocols that they are using, the port numbers that they are listening on, the RPC version that they are running, and the version of the particular service that they are offering. This is a common method for attackers to gather information about RPC programs on your systems, but it's also an excellent way for an auditor to determine through testing what's actually running on the box.

© 2016 David Hoelzer & Enclave Forensics™

UNIX Services

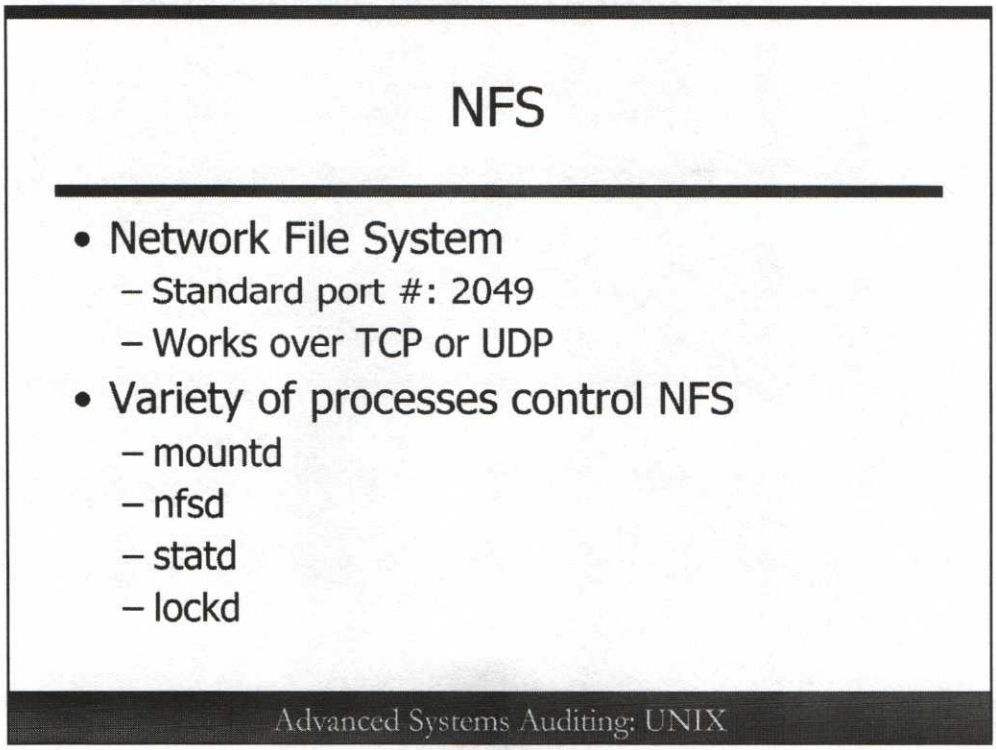
- Standard Services:
  - Portmapper and RPCs
  - NFS
  - X, CDE, and Friends



Advanced Systems Auditing: UNIX

The next standard service that we discuss is the NFS service.

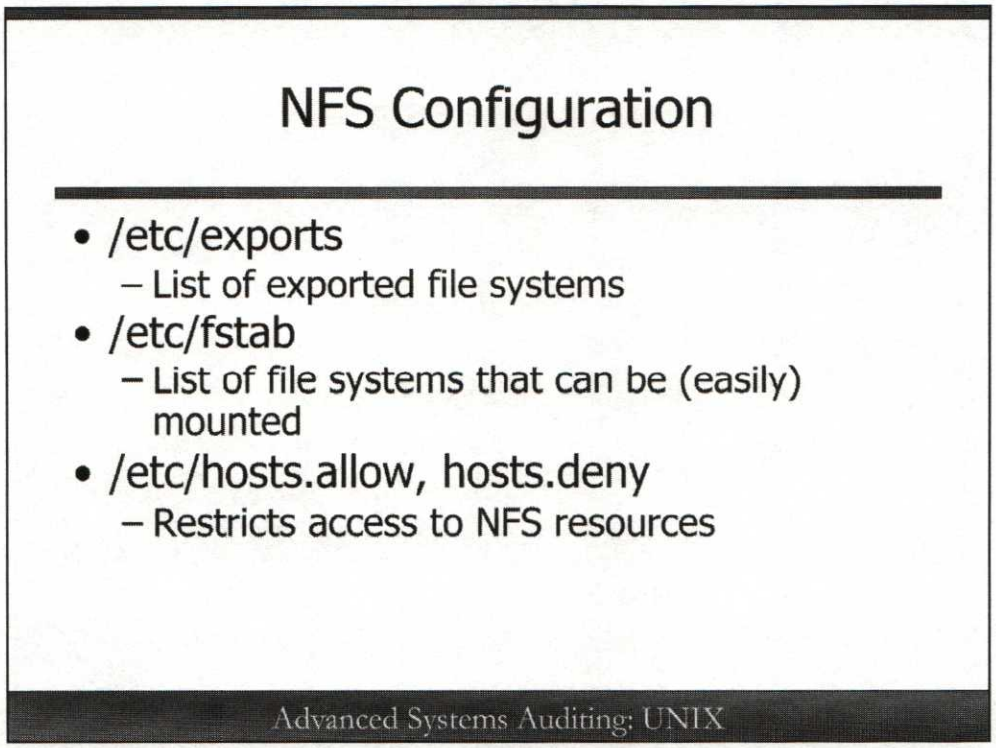
© 2016 David Hoelzer & Enclave Forensics™



Network File System (NFS) has been in existence for more than a decade and was created to provide a standard mechanism for sharing file and disk resources across a network between UNIX systems. NFS is such a well-defined standard that it can be used to share file systems between all varieties of UNIX systems, regardless of the actual architecture of the system or the format of the file systems in question. This creates a tremendous advantage when it comes to creating a simple way to move information around an organization.

NFS prefers to bind or listen on port number 2049 and has the capability of functioning over either TCP or UDP. NFS is a rather intricate system and requires a variety of services in order to function properly. These include 'mountd,' 'nfsd,' 'statd,' and 'lockd.'

© 2016 David Hoelzer & Enclave Forensics™



A few files to know about that control the configuration of the NFS services are '/etc/exports,' '/etc/fstab,' '/etc/hosts.allow,' and '/etc/hosts.deny.'

The '/etc/exports' file is used to enumerate the file systems on the local machine that are being exported for access across the network.

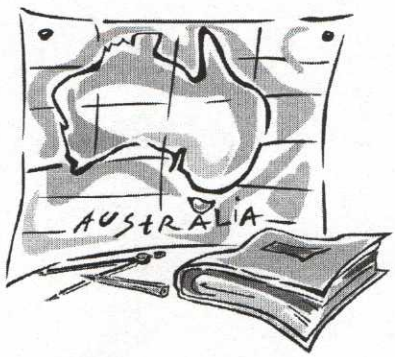
'/etc/fstab' is used to enumerate the "known" file systems that can be easily mounted. This does not mean that other file systems cannot be mounted, but these can be mounted using what amounts to an alias because all of the other information about the file system location and type as well as the mount point are defined in the 'fstab' or file system table.

The '/etc/hosts.allow' and '/etc/hosts.deny' can be used to control access to network-based services. The use of these access control files with RPCs and NFS is dependent on using a modern RPC service and portmapper. Older versions of portmapper have no ability to reference the hosts.allow or hosts.deny.

© 2016 David Hoelzer & Enclave Forensics™

UNIX Services

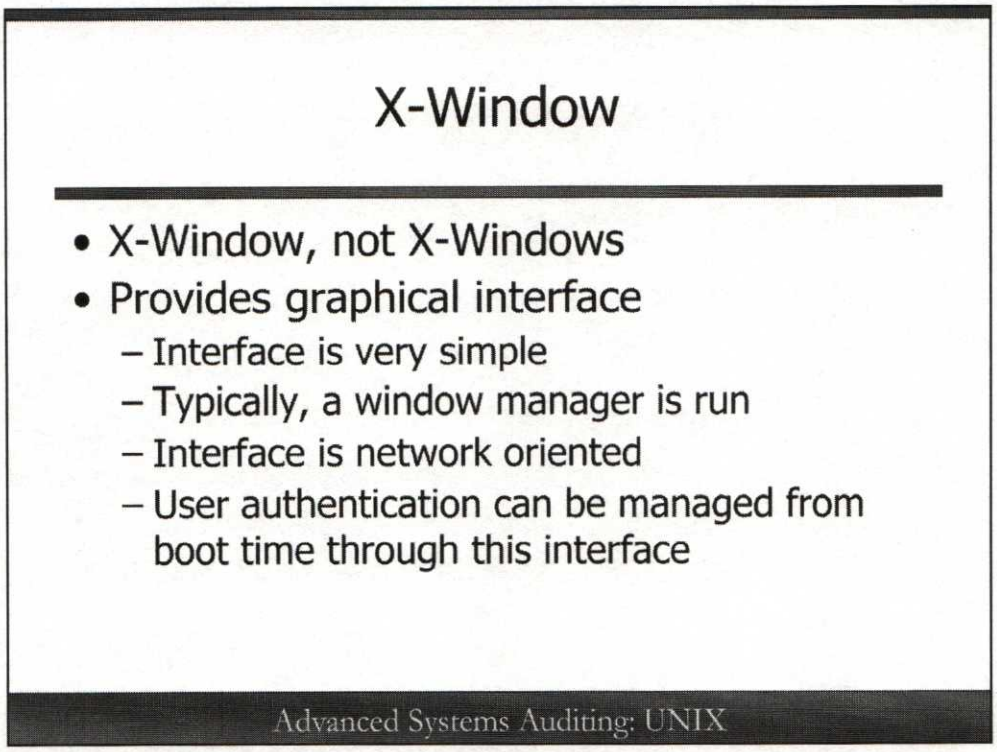
- Standard Services:
  - Portmapper and RPCs
  - NFS
  - X, CDE, and Friends



Advanced Systems Auditing: UNIX

The last common service that we cover at this point is the X-Window system and supporting components.

© 2016 David Hoelzer & Enclave Forensics™



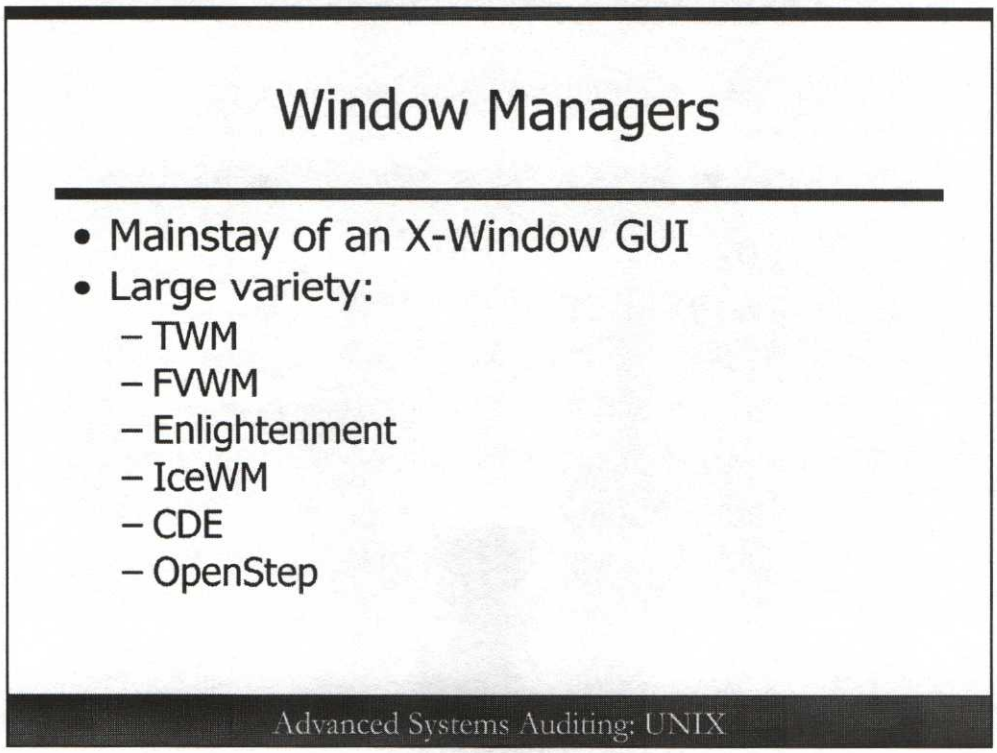
To satisfy the UNIX purists, we should first point out that the graphical system that we are considering is not "X-Windows"; the proper name of the system is "X-Window." With that out of the way, there's a good chance that we might lapse into calling it X-Windows from time to time.

The X-Window system does not provide much more than a full-screen interface out of the box. Most of the familiar interface components that you might have used are actually usually a function of the window manager that you choose to run. We talk more about the window managers in a moment.

The X-Window system is also network oriented. Not only is it possible to connect to the window system across the network or to send a window to a remote system across the network, but the system actually uses the loopback interface on the local system in order to produce windows. This turns out to be a pretty powerful feature because it allows you to run an application on one machine while the graphical output goes to another machine anywhere in the world!

The X-Window system also provides for a graphical authentication system using the 'XDM' program (X Display Manager). There are a variety of other third-party display managers as well, of which 'GDM' (Gnome Display Manager) is one of the best known.

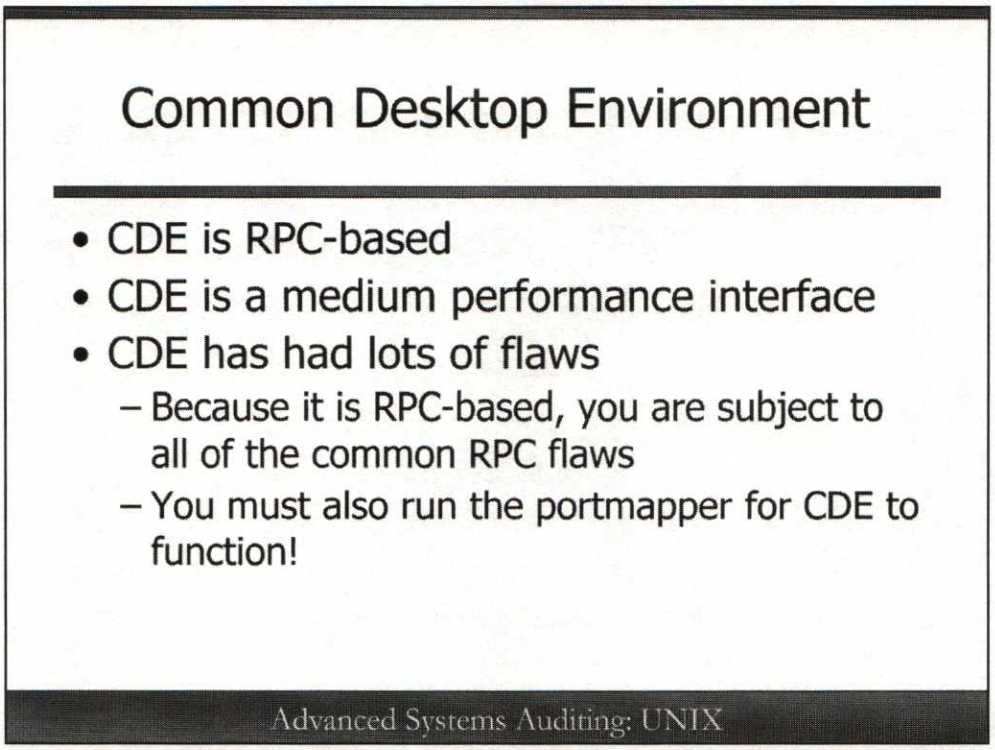
© 2016 David Hoelzer & Enclave Forensics™



The X-Window system itself gives you only the most basic window functionality. To actually move windows around, create menus, manage multiple screens, or any of the other most common GUI functions, you must run a window manager of some sort. Window managers come in all shapes and sizes, each focused on some particular aspect of the entire GUI experience. For instance, if you're looking for really fancy-looking windows and backgrounds, enlightenment is for you. If you're looking for very simple and efficient, you might want to try out Tom's Window Manager (TWM).

Most commercial UNIX systems provide a standard window manager or GUI environment known as "Common Desktop Environment (CED)".

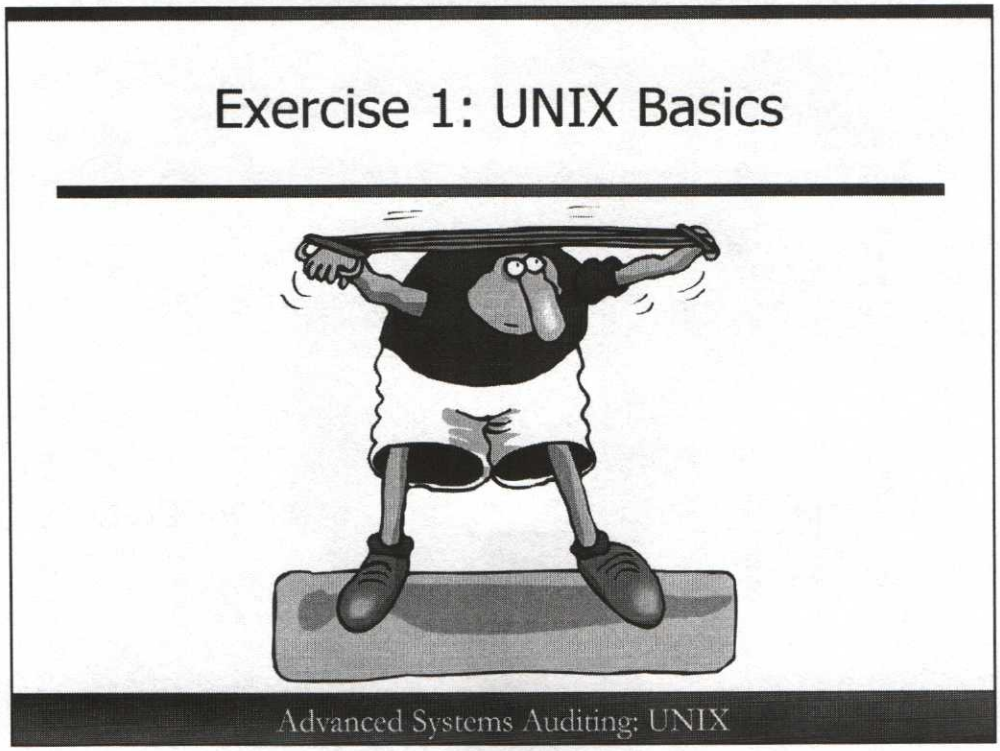
© 2016 David Hoelzer & Enclave Forensics™



CDE is an RPC-based GUI system that provides a great deal of functionality to the user. It includes icon trays, a quick start bar, virtual desktops, and more. Most of the system is implemented using RPCs in order to divorce the actual functionality from the GUI itself. One of the reasons why this sort of model is used is to allow you to easily port the GUI itself from one platform to another. The only modification that typically will need to be made is with the back-end functions, which are the RPCs.

The big drawback to running an interface like CDE is that you must run the portmapper for it to function. You also naturally inherit all of the issues that come along with RPC programs. CDE has had quite a few remotely exploitable vulnerabilities in its history, so you might want to consider carefully whether or not it might be time to migrate to some other window manager.

© 2016 David Hoelzer & Enclave Forensics™



Please turn to Exercise 1 in your workbook and begin the exercise. If you are taking this class online, pause the audio while you experience the exercise.

© 2016 David Hoelzer & Enclave Forensics™