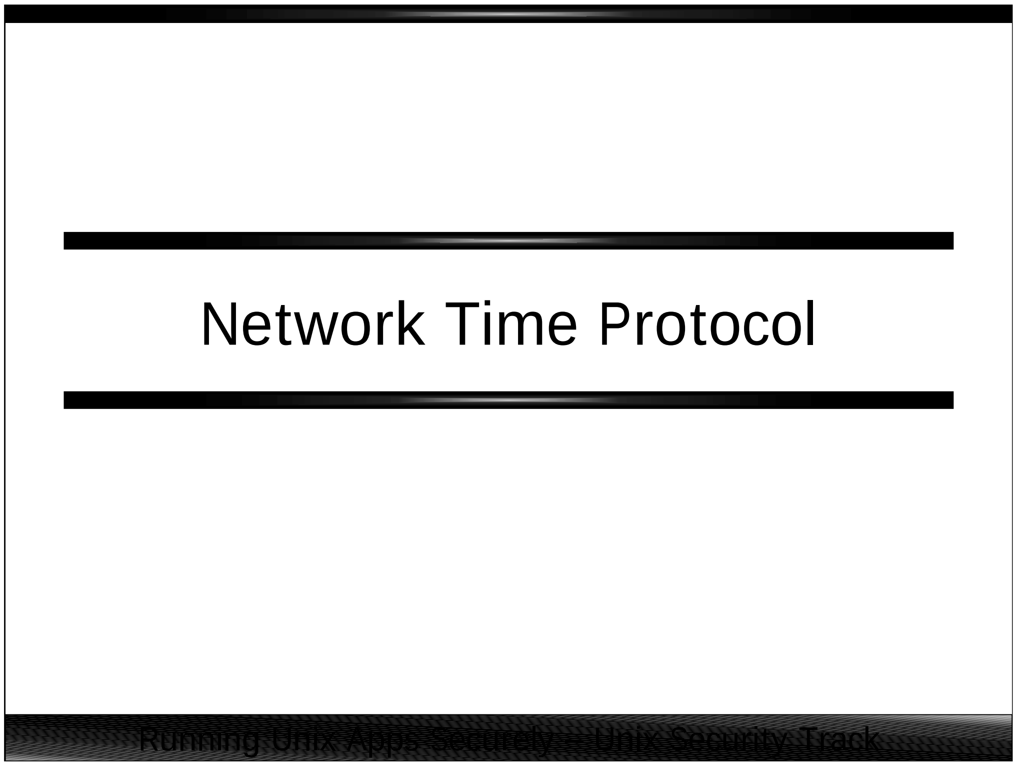


Network Time Protocol

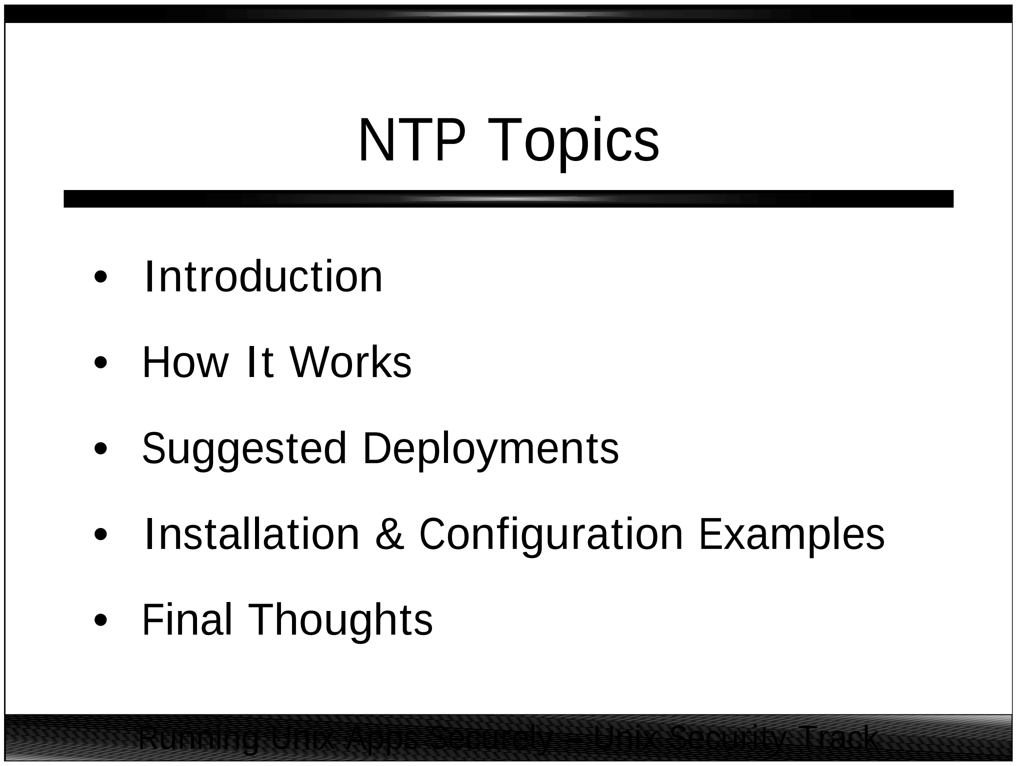
Running Unix Apps Securely -- Unix Security Track

Hal Pomeranz * Founder/CEO * *hal@deer-run.com*

Deer Run Associates * PO Box 20370 * Oakland, CA 94620-0370

+1 510-339-7740 (voice) * +1 510-339-3941 (fax)

http://www.deer-run.com/



NTP Topics

- Introduction
- How It Works
- Suggested Deployments
- Installation & Configuration Examples
- Final Thoughts

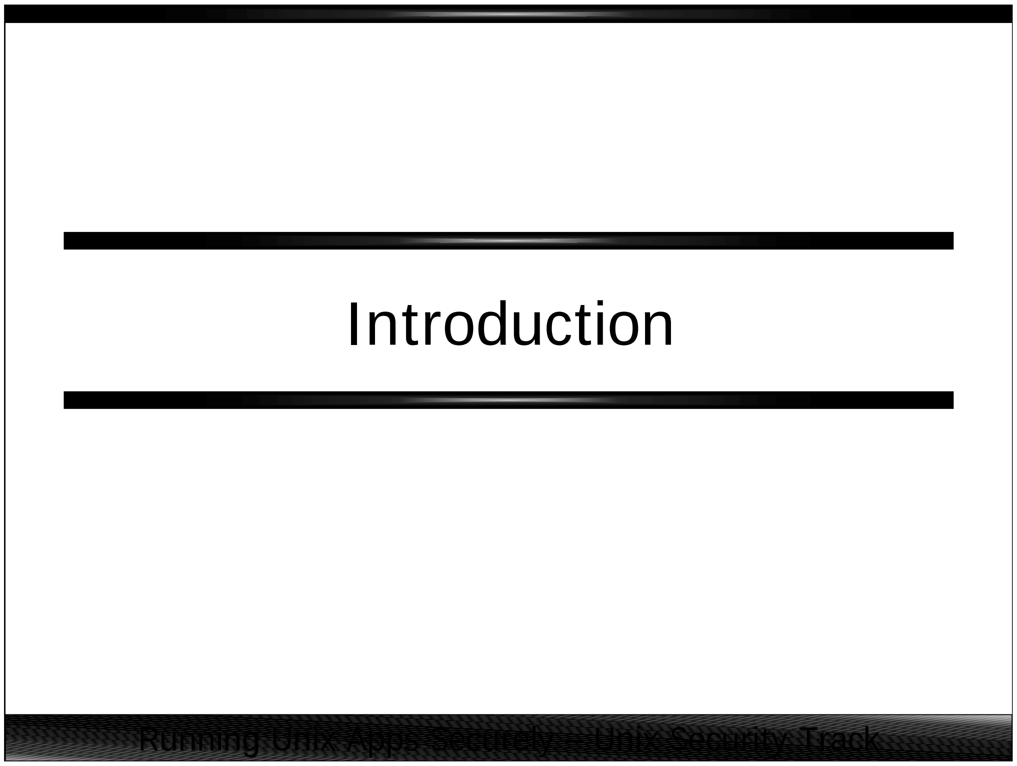
Introduction is a general overview of the Network Time Protocol including some historical perspectives.

How It Works provides insight on how NTP runs in the Internet environment and introduces some terminology which will be used in the rest of the course.

Suggested Deployments covers high-level architectural issues common to most enterprise NTP configurations.

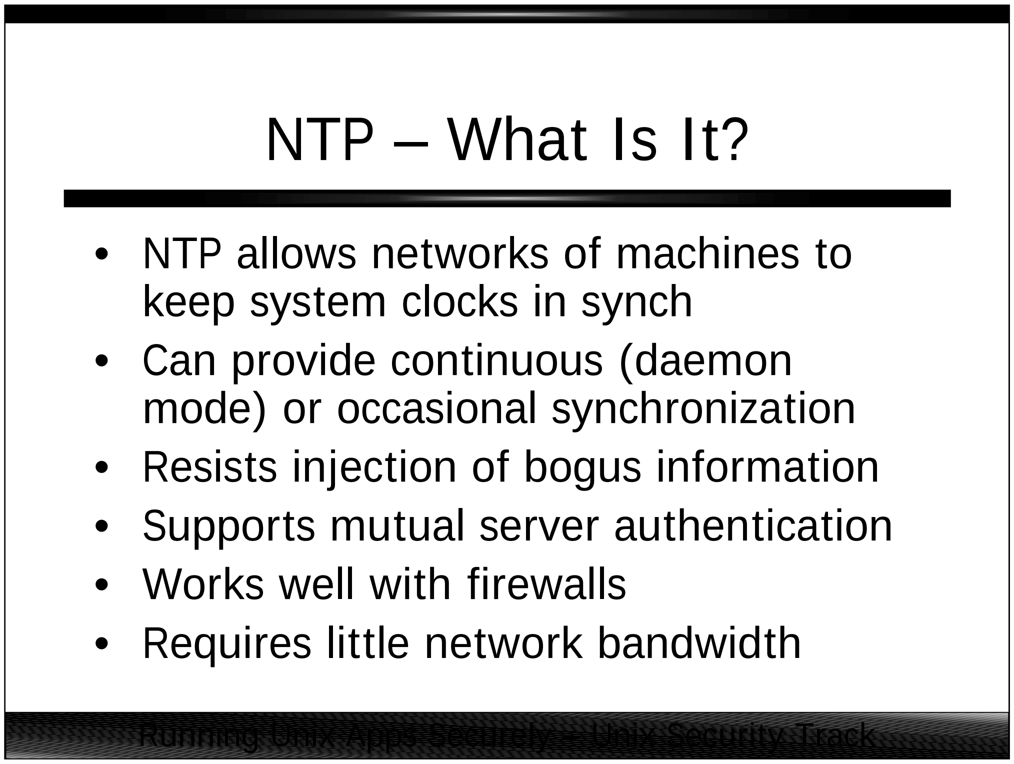
Installation and Configuration Examples includes not only actual NTP configuration files but also provides information on obtaining and building the Open Source version of NTP.

Final Thoughts includes some parting shots and some useful URLs.



Introduction

Introduction is a general overview of the Network Time Protocol including some historical perspectives.



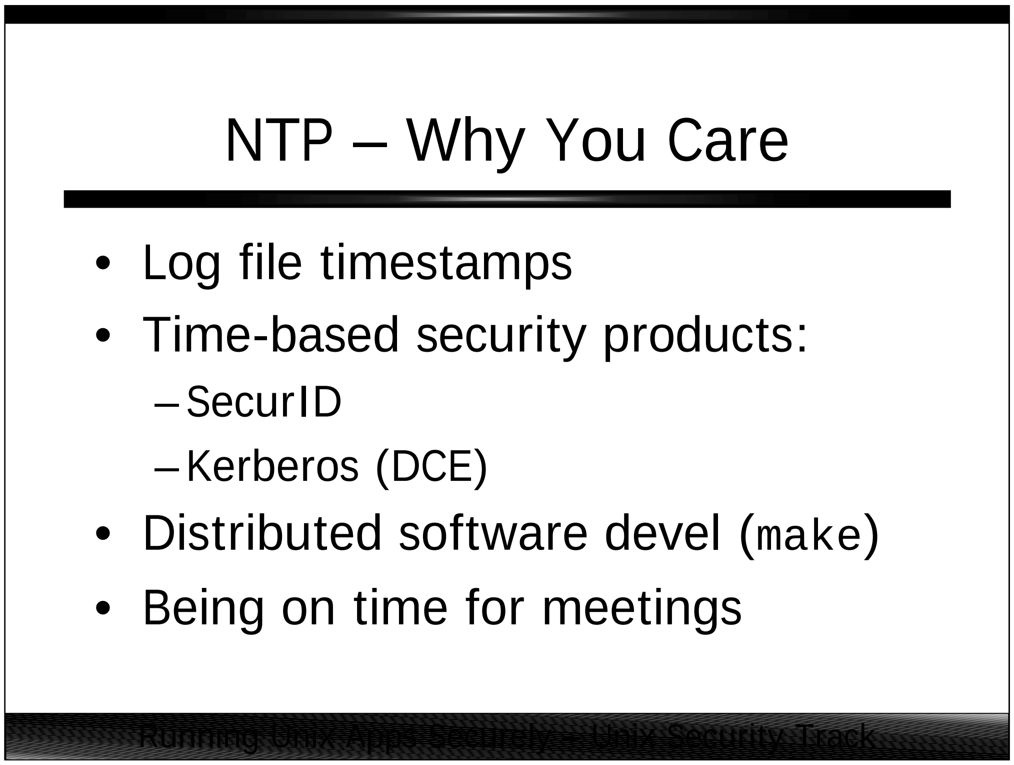
NTP – What Is It?

- NTP allows networks of machines to keep system clocks in synch
- Can provide continuous (daemon mode) or occasional synchronization
- Resists injection of bogus information
- Supports mutual server authentication
- Works well with firewalls
- Requires little network bandwidth

NTP is the Internet standard time synchronization protocol. The primary author and maintainer is David Mills at the University of Delaware. The NTP code has been ported to a bewildering variety of Unix and non-Unix machines and is rigorously backwards compatible.

NTP can be run as a daemon which regularly polls a group of time servers and keeps the system clock in synch from moment to moment. Accuracy to within hundredths of a second is standard. The NTP distribution also comes with an `ntpdate` program (similar to the BSD `rdate` program) which can be used on client machines at boot time or called every few hours out of `cron` to keep less critical machines in rough synchronization with the rest of the organization (a similar tool for Win32 machines is available for free from `http://www.thinkman.com/dimension4/index.html`).

When run in daemon mode, NTP is essentially its own proxy. The usual configuration is to have the hosts on your external network synch with accurate clock sources on the Internet and then have your internal hosts synchronize off the hosts on your external network.



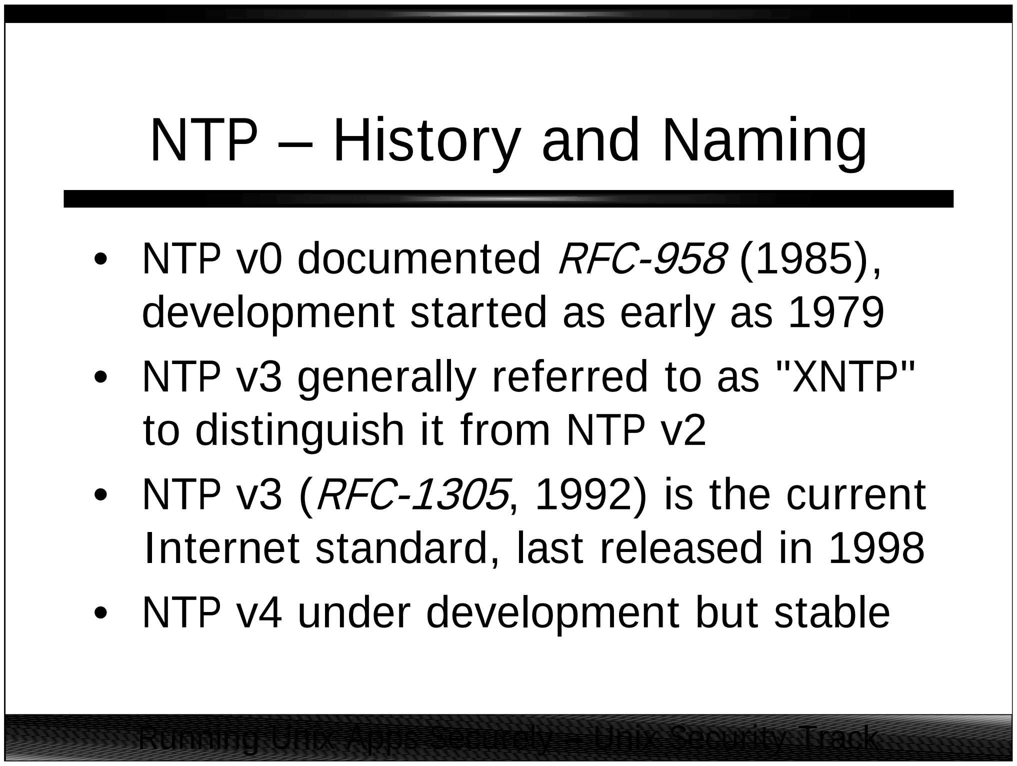
NTP – Why You Care

- Log file timestamps
- Time-based security products:
  - SecurID
  - Kerberos (DCE)
- Distributed software devel (`make`)
- Being on time for meetings

Time synchronization is vitally important to your organization.

From a security perspective, effective prosecution of security incidents requires accurate ***matching*** timestamps on all log files. Any discrepancies will complicate or sabotage legal proceedings. There is also a reasonably large body of security software which requires accurate time information to work effectively.

If you are a software development organization, correct time information across your NFS servers and clients can make or break your development-- particularly if you use a parallel/distributed `make` product.



NTP – History and Naming

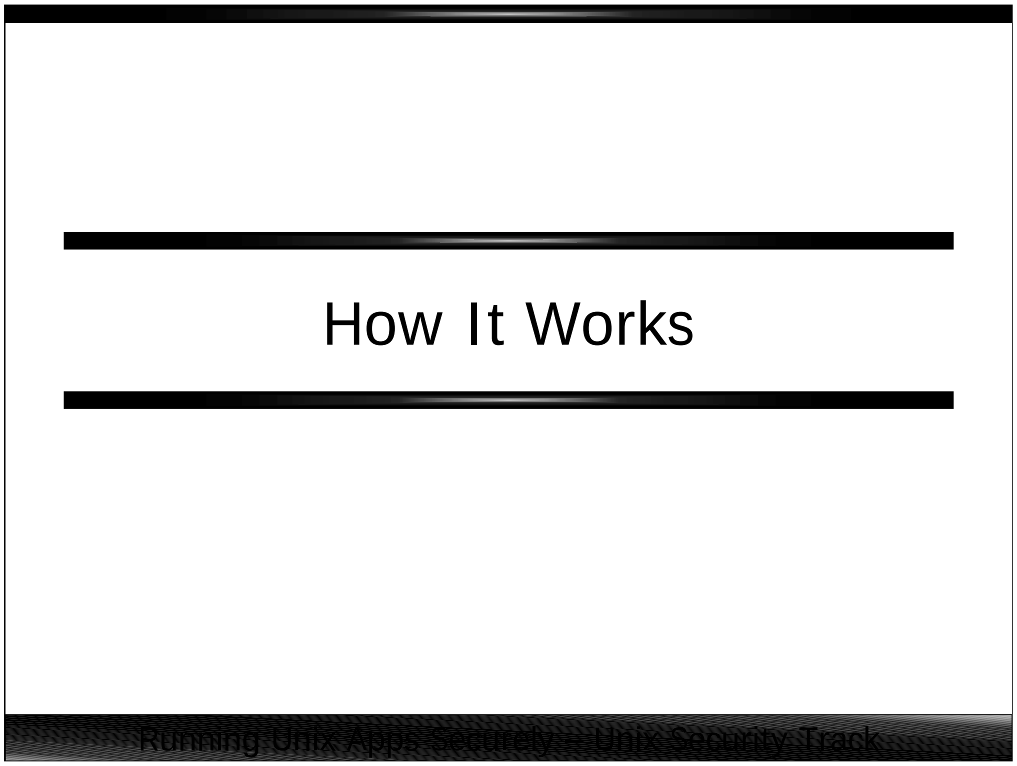
- NTP v0 documented *RFC-958* (1985), development started as early as 1979
- NTP v3 generally referred to as "XNTP" to distinguish it from NTP v2
- NTP v3 (*RFC-1305*, 1992) is the current Internet standard, last released in 1998
- NTP v4 under development but stable

Running Unix Apps Securely – Unix Security Track

NTP v3 is the current Internet standard version of the protocol. When initially released in 1992, NTP v2 was still in heavy use particularly on PC platforms. The v3 release was commonly called "XNTP" to distinguish it from v2 implementations. NTP v4 has gone back to just "NTP".

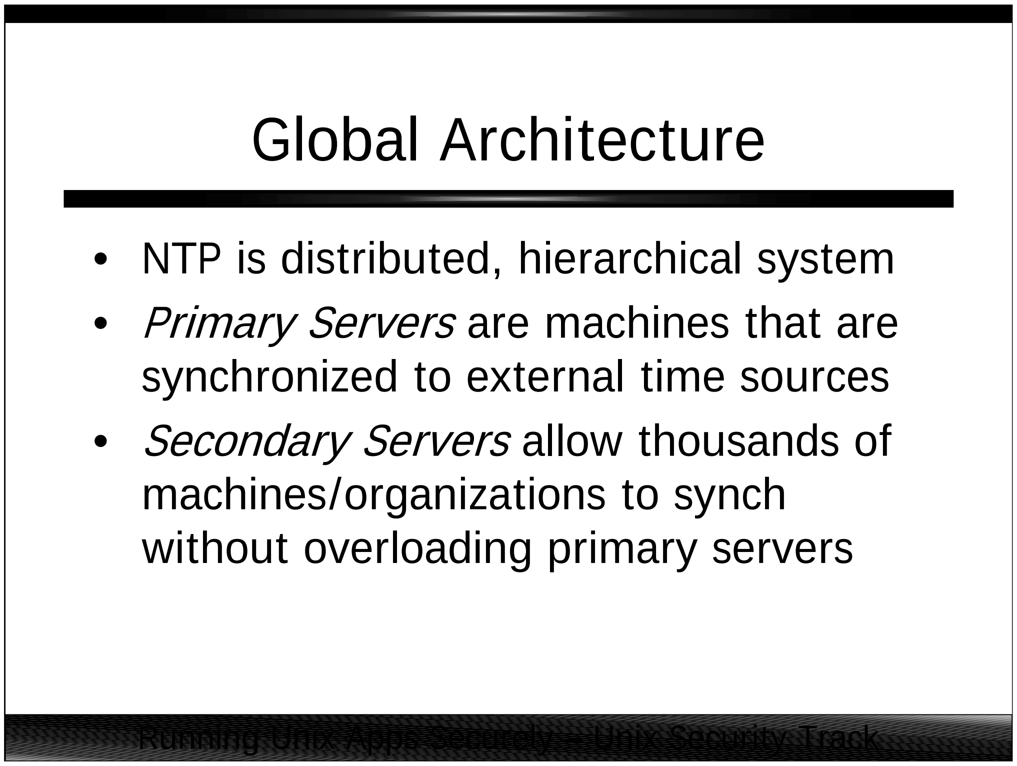
More info from a personal message from David Mills (5/2/2000):

> "NTP version 0-4 were/are real and distinct. See rfc1305 appendix [D] for historic conformance statement. Versions 1, 2 and 3 were documented in rfcs; version 4 is still wet. The reference implementations for 0-3 were on PDP11 fuzzballs. Later reference implementations for 3, 4 are in Unix. …
>
> The original implementation which evolved to NTP was in the Hello routing algorithm used in the fuzzballs from 1979. It was documented in an early Internet Experiment Note circa 1981-3. I don't remember the exact rollout dates, but that doesn't matter much, since the architecture, protocol and algorithms evolved essentially in a continuous manner to the present day."



How It Works

How It Works provides insight on how NTP runs in the Internet environment and introduces some terminology which will be used in the rest of the course.



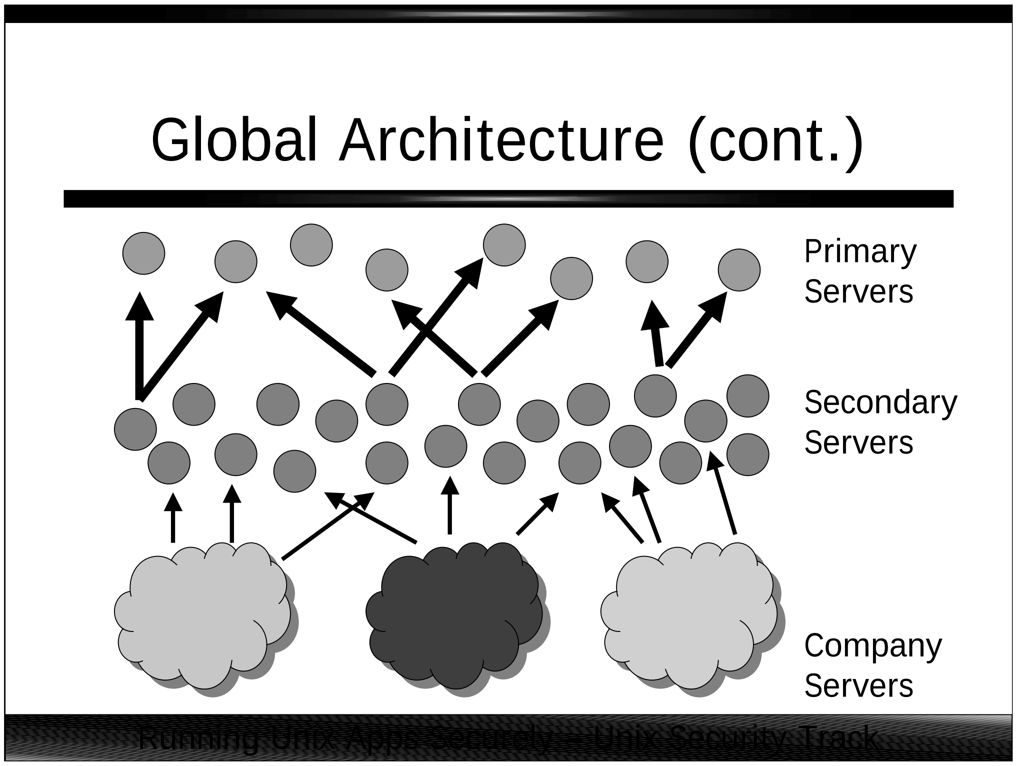
Global Architecture

- NTP is distributed, hierarchical system
- *Primary Servers* are machines that are synchronized to external time sources
- *Secondary Servers* allow thousands of machines/organizations to synch without overloading primary servers

Running Unix Apps Securely – Unix Security Track

Like many other Internet-based distributed system, NTP is hierarchically-oriented. That is, there is a small core of *primary time servers* who set their clocks against external, highly accurate sources of time information (Cesium clocks, GPS receivers, etc.). Below this level are *secondary servers* which are responsible for distributing time from the primary servers to the rest of the Internet. Secondary servers are required because, while a given NTP server can service hundreds (if not thousands) of other time servers, the number of machines requiring time synchronization on the Internet today numbers into the millions of machines.

As we will see later in the course, this hierarchical structure is continued as the NTP infrastructure permeates the individual organizations which are connected to the Internet– that is each organization synchronizes several tertiary servers to the publicly available secondary servers, and then proceeds to synchronize other hosts in their enterprise against these local tertiary machines.

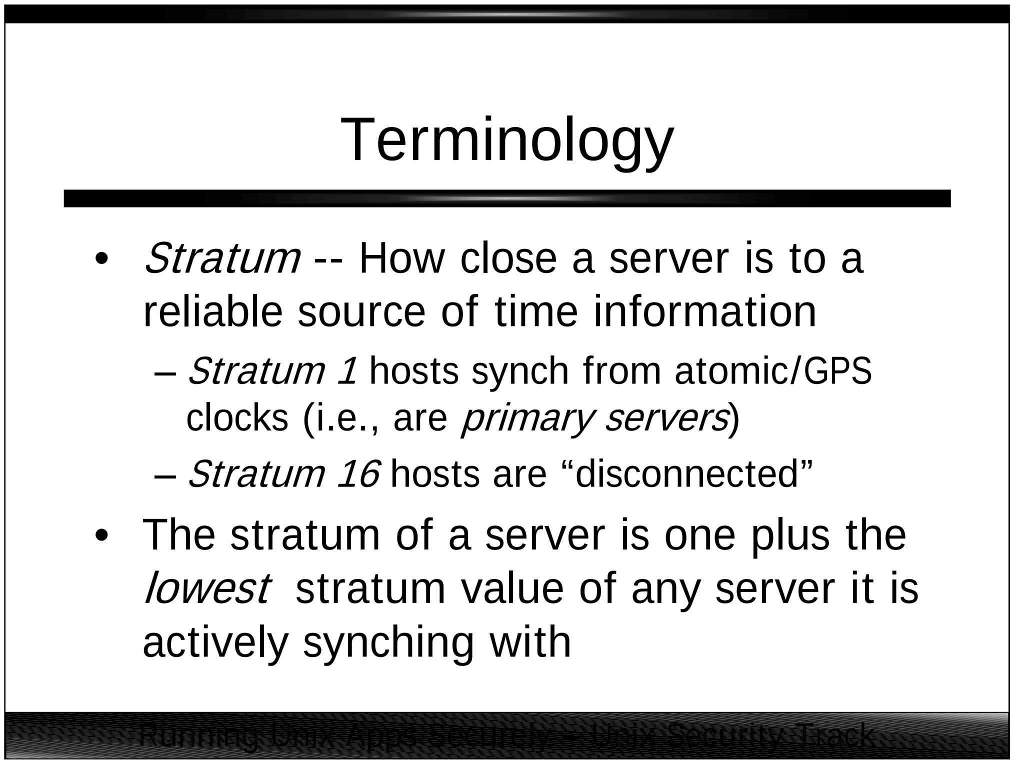


Global Architecture (cont.)

Pictorially speaking, the Internet time hierarchy looks something like what is shown above.

At the top level are the relatively few (somewhere between 200 and 300) "publicly available" primary servers. Each of these machines shares time information with several dozen secondary servers– there is some overlap, but there must be thousands of publicly available secondary servers on the Internet today.

Below the secondary server level are all of the organizations with direct Internet connections. These organizations synchronize off the secondary server tier.

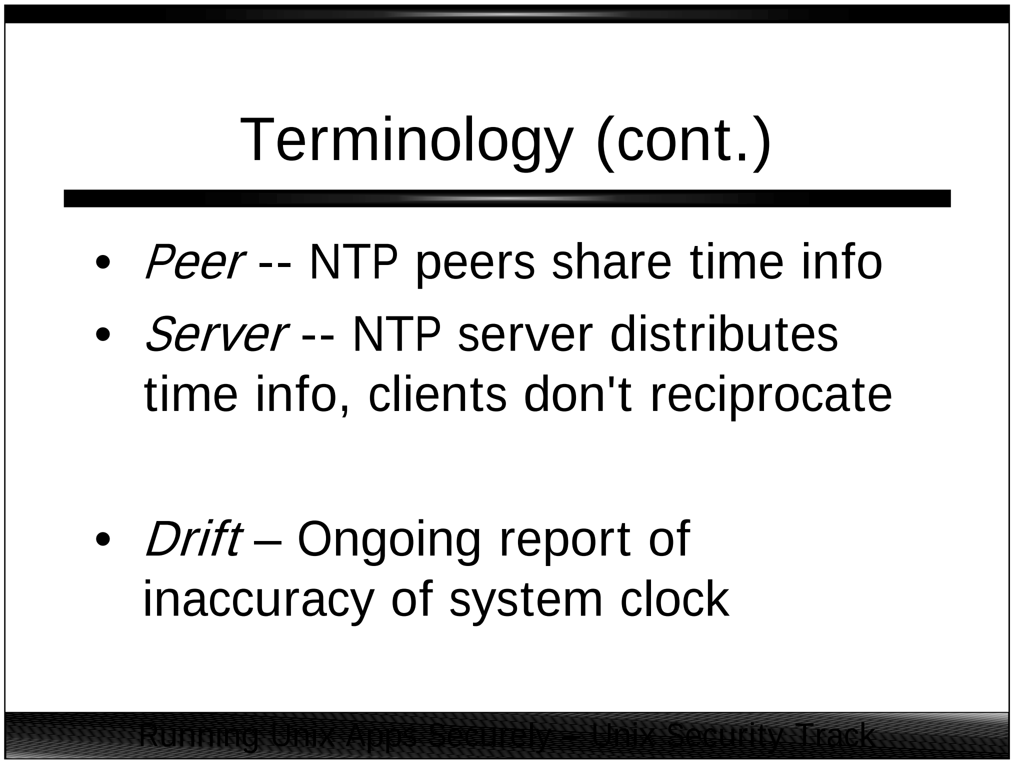


Terminology

- *Stratum* -- How close a server is to a reliable source of time information
  - *Stratum 1* hosts synch from atomic/GPS clocks (i.e., are *primary servers*)
  - *Stratum 16* hosts are “disconnected”
- The stratum of a server is one plus the *lowest* stratum value of any server it is actively synching with

In other words, if your host is synching against one stratum 1 server and two stratum 2 servers, then your host is a stratum 2 server. Stratum values are dynamic-- if you suddenly lose connectivity to that stratum 1 server, then your stratum value will drop to 3.

Generally speaking, hosts will prefer time synchronization information from lower stratum hosts.



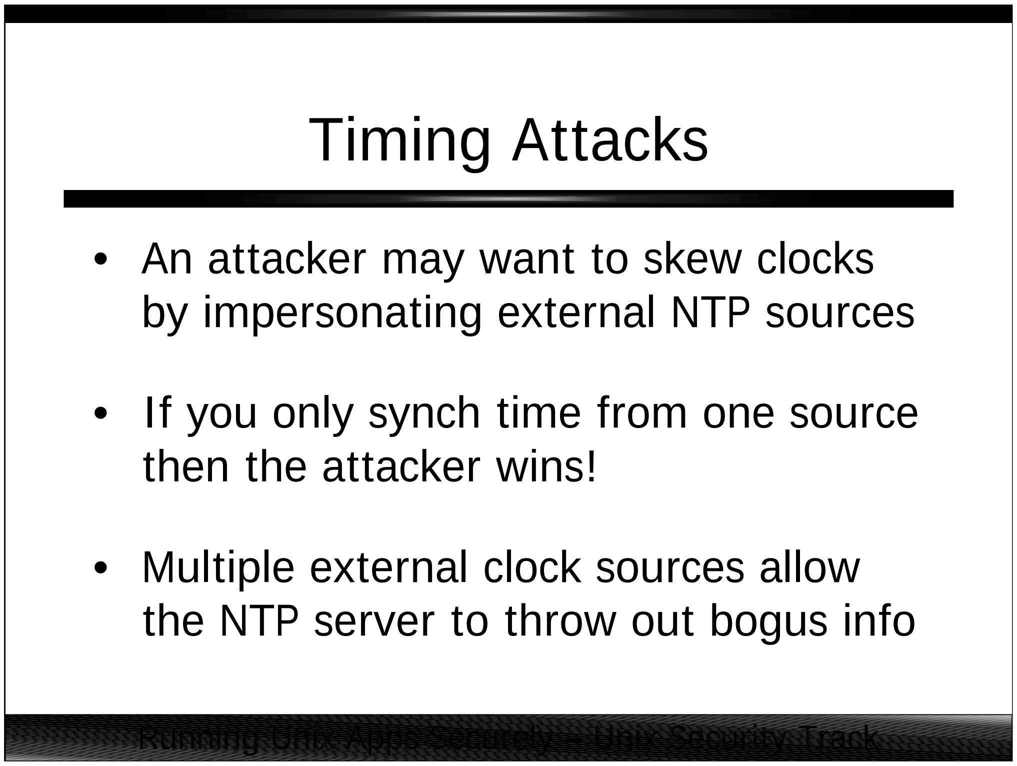
Terminology (cont.)

- *Peer* -- NTP peers share time info
- *Server* -- NTP server distributes time info, clients don't reciprocate
- *Drift* – Ongoing report of inaccuracy of system clock

NTP servers can either be configured in a peer-to-peer relationship or in a master/slave relationship.

A peer relationship implies that both parties are aware of each other's existence and have the other machine configured in their local NTP configuration file. However, in a master/slave relationship only the client configuration file needs to contain the server information, so the server may be unaware of the client's existence until the client actually requests time information from the server ("pull" rather than "push"). Clients who take time synchronization information from a server without notifying that server's administrator are generally referred to as *clock suckers*.

`xntpd` keeps moment to moment statistics on how much average variance the local clock has from the time standard. This *drift* value is used to keep the system clock accurate in the event of a network partition that causes the host to lose time synch information. Once the drift value has been accurately estimated, `xntpd` is capable of keeping VERY accurate time even without external clock information.

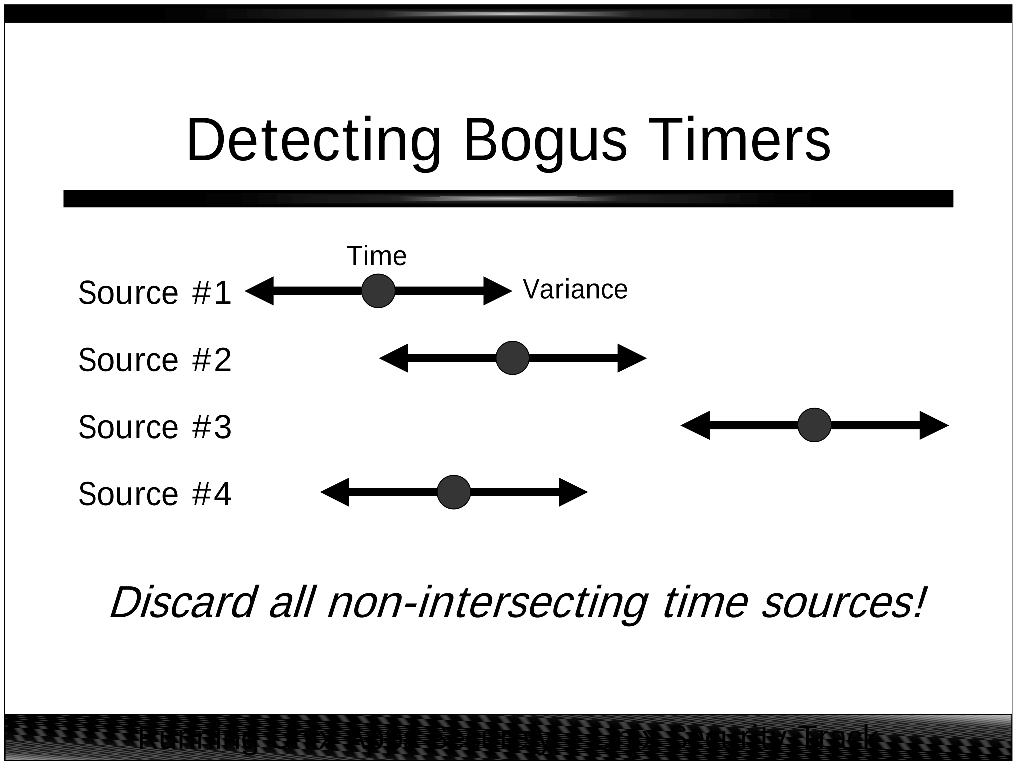


Timing Attacks

- An attacker may want to skew clocks by impersonating external NTP sources
- If you only synch time from one source then the attacker wins!
- Multiple external clock sources allow the NTP server to throw out bogus info

One of the problems with your security depending upon time information from an external source is that a knowledgeable attacker may try to impersonate your external time source and skew your machine's clock. Success may allow the attacker to replay time-based security credentials against your infrastructure.

NTP has built-in algorithms for detecting and ignoring obviously bogus time information. For these to be effective, however, your server must be receiving updates from more than one external clock source…



Detecting Bogus Timers

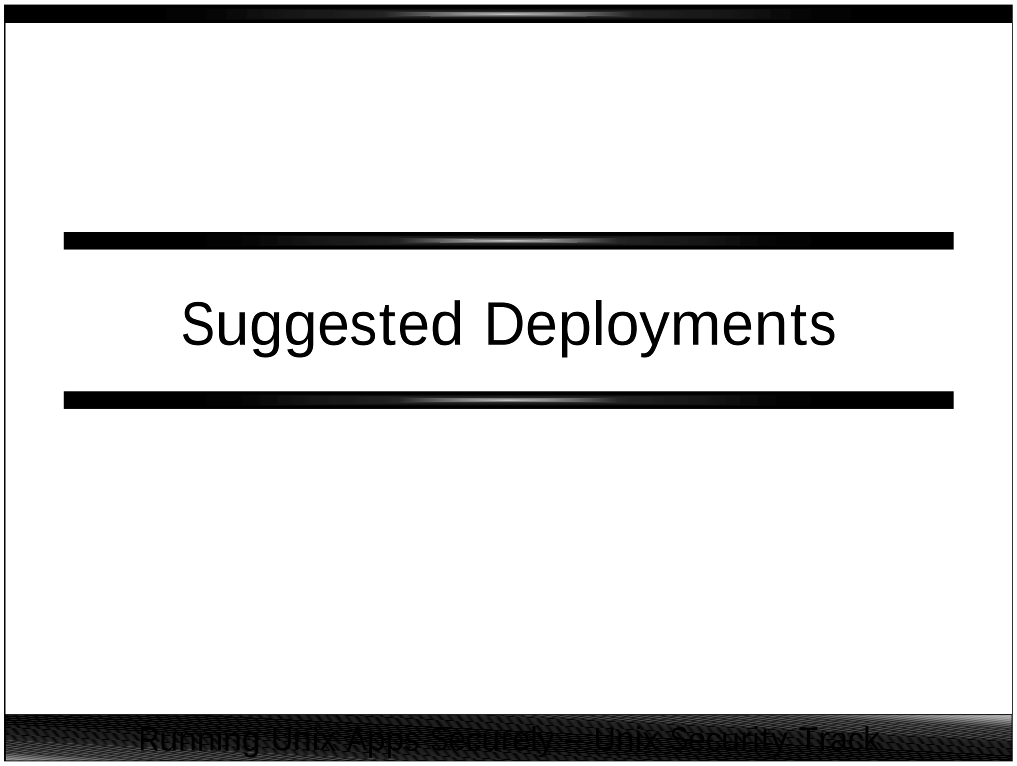
Discard all non-intersecting time sources!

Running Unix Apps Securely – Unix Security Track

Let's say your local NTP server is getting time information from four external sources. Each source reports their notion of the current time (represented by the dots in the picture above) and your local NTP server is able to calculate a maximal error threshold for each time source, which gives a small range of potentially "correct" time values (represented by the arrowed lines).

It turns out that the actual time must lie somewhere in the intersection of the ranges reported by the various time sources. Your local NTP server searches for the largest group of intersecting time ranges from all of its potential time sources and then sets the time based on the information from those intersecting servers.

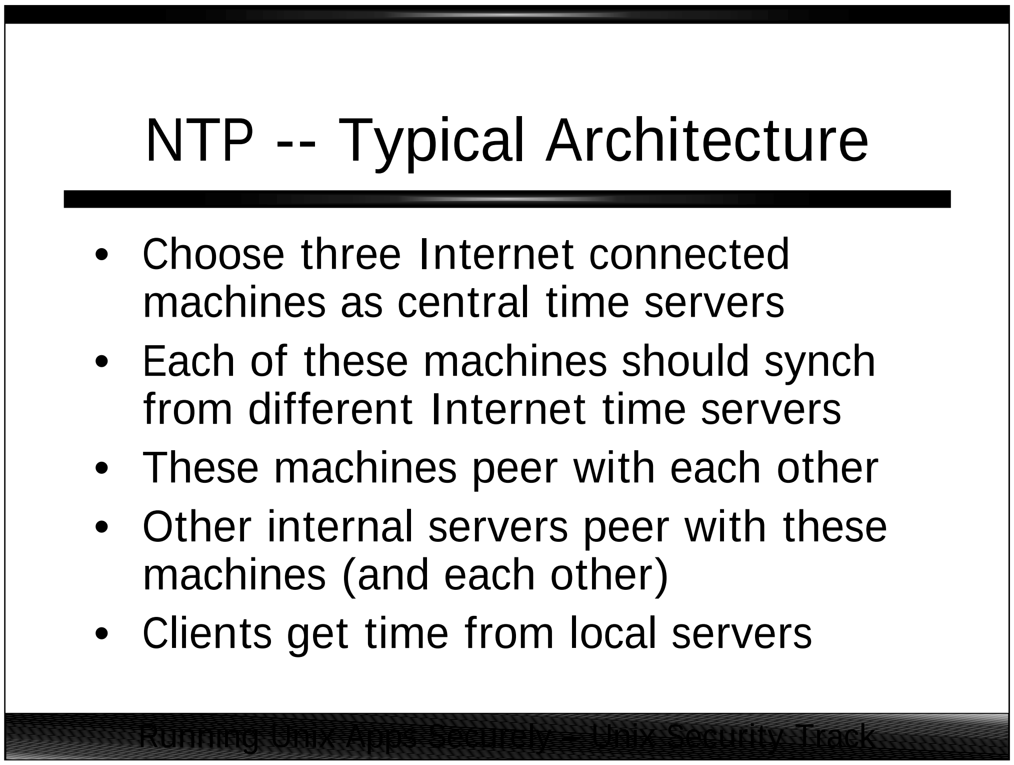
In the example shown above, sources #1, #2, and #4 all intersect to varying degrees so these are valid clock sources (*true chimers*). Source #3 is reporting time outside of the range of any of the other servers, so it is invalid (a *false ticker*)– this may be because source #3 has been compromised by an attacker, or it may just be that source #3 has a terrible system clock or is getting time from some other wildly inaccurate time source.



Suggested Deployments

Running Unix Apps Securely -- Unix Security Track

Suggested Deployments covers high-level architectural issues common to most enterprise NTP configurations.



NTP -- Typical Architecture

- Choose three Internet connected machines as central time servers
- Each of these machines should synch from different Internet time servers
- These machines peer with each other
- Other internal servers peer with these machines (and each other)
- Clients get time from local servers

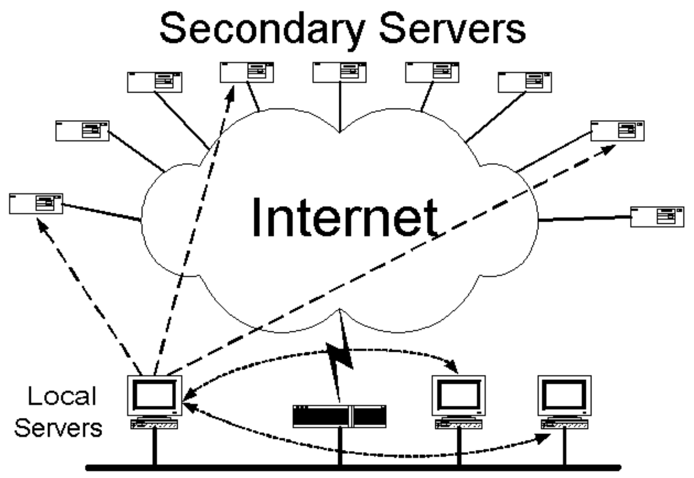
Running Unix Apps Securely -- Unix Security Track

Again we see the hierarchical nature of good NTP architectural design. The idea is to avoid overloading low stratum servers and to keep hosts from having to leave their local LANs for time synchronization information. This is particularly important over WAN links.

Desktop clients should probably never `peer` with other machines, and it may be sufficient to simply use `ntpdate` to keep desktop machines in synch.

A list of accurate Internet clock sources is provided on the NTP Web site (URL at the end of this course).

External Time Synch Arch

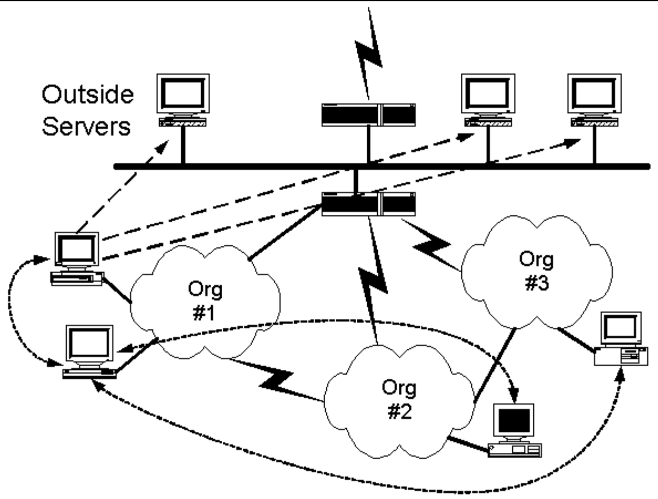


Here's a simple diagram showing how an organization with a single Internet feed might get good external time information from the Internet.

Three (or possibly more) local servers are deployed on some externally connected network. Each of these machines is synchronized against a least three external secondary servers (there are three local servers in the picture above, so nine external secondary servers are required). Each of the local servers also peers with the other two local machines.

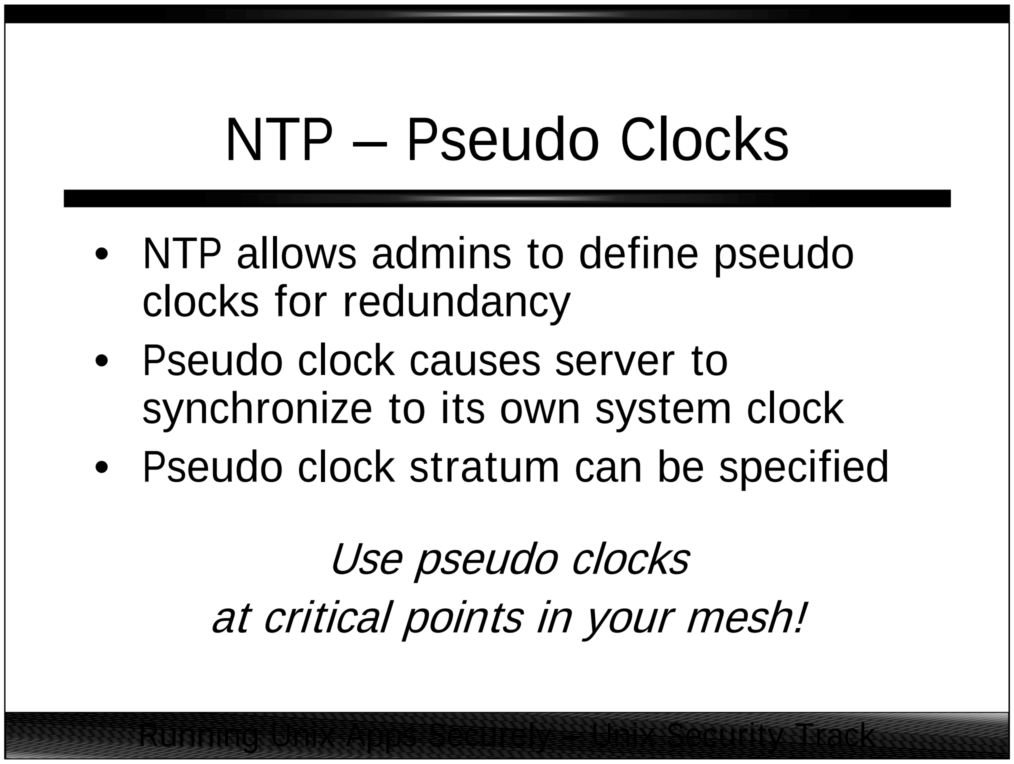
Note that not all lines representing the complete set of master/slave and peer-to-peer relationships are drawn in the diagram above– this was done in order to make the picture less cluttered.

Internal Time Synch Arch



Now let's suppose that this same organization happens to have three distinct organizational units with various LAN and WAN connections between them. Each of these organizations also has one or more local NTP servers.

Each of the NTP servers in each of the different organizational units will peer with all of the other internal NTP servers. These internal servers will also peer with the servers which are getting time information from the external Internet. For large organizations, you may wish to configure this as a master/slave relationship rather than peer-to-peer.



NTP – Pseudo Clocks

- NTP allows admins to define pseudo clocks for redundancy
- Pseudo clock causes server to synchronize to its own system clock
- Pseudo clock stratum can be specified

Use pseudo clocks at critical points in your mesh!

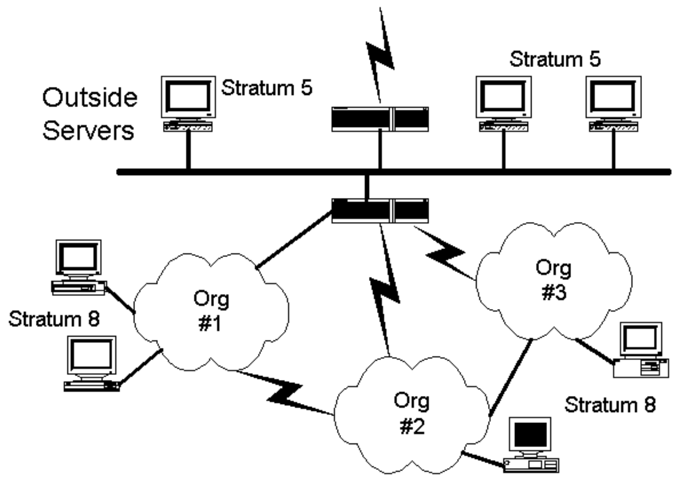
Running Unix Apps Securely – Unix Security Track

There will come a time when your Internet gateway or other WAN link will go down and all of your local time servers will suddenly drop to stratum 16 and their clients will stop listening to time synchronization information. A few carefully located pseudo clocks will prevent this from happening.

Generally speaking, each of your locations should have at least one master time source with a pseudo-clock definition. Having several pseudo-clocks per site will prevent a single machine with an inaccurate system clock from distorting your network time.

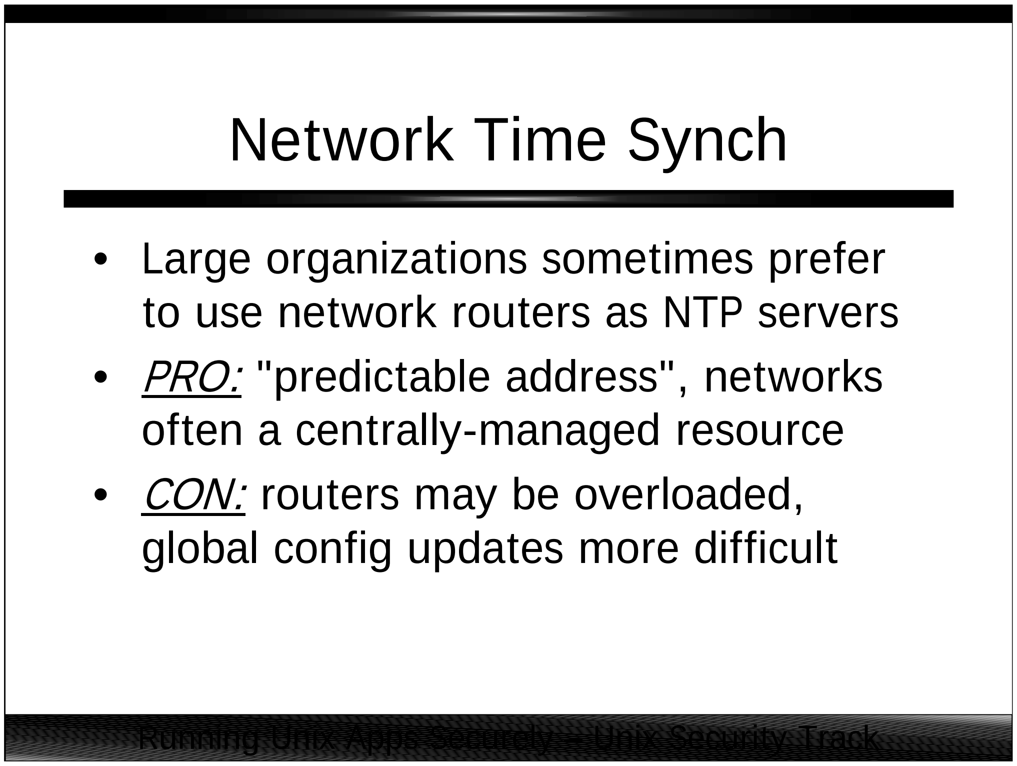
You must be careful, however, to set the stratum of your pseudo-clock at least a couple of points higher than the stratum that the server usually operates at– this prevents the pseudo-clock from interfering with the information that you're acquiring from the Internet. This means that pseudo-clocks should generally be configured at strata 5-8.

Internal Time Synch Redux



Here's our internal network architecture diagram again, this time showing appropriate stratum values for the various internal and external pseudo-clocks. The pseudo-clocks on the "Outside Servers" should be configured at stratum 5– this means that if the Internet connection becomes severed then the "Outside Servers" will become ***stratum 6*** servers (the stratum of the pseudo-clock *plus one*).

This implies that the internal time servers will drop to stratum 7 when the Internet link is lost. Therefore, configure the pseudo-clocks on these internal machines at stratum 8 so that the internal machines will only synchronize off of these clocks if they lose their connectivity to the rest of the company.



Network Time Synch

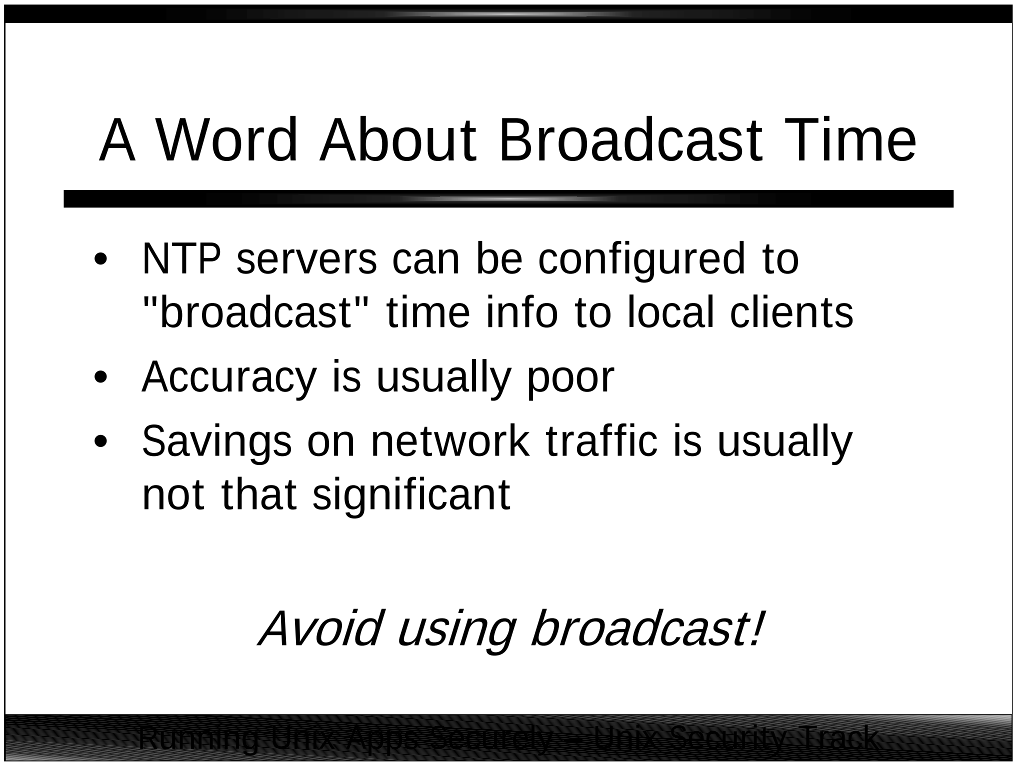
- Large organizations sometimes prefer to use network routers as NTP servers
- *PRO:* "predictable address", networks often a centrally-managed resource
- *CON:* routers may be overloaded, global config updates more difficult

Running Unix Apps Securely -- Unix Security Track

Most almost every router on the market today is capable of being an NTP server. Many organizations choose to simply use their router infrastructure to supply time to their entire enterprise.

This often works well because network administration is usually a centralized corporate function and so time synchronization can be provided reasonably seamlessly throughout the entire organization. Also routers typically appear at a "predictable" address on a given LAN (often the `.1` or `.254` address) and so machines can build their own NTP configuration files "on the fly" at boot time to simply synchronize against their default router.

On the minus side, your routers may already be too overloaded to be providing timing information to all of the locally attached hosts. The router also becomes an even larger single point of failure for your LAN. Also, it's typically easier to do global updates to host-based configuration files (using tools like `rdist` and `rsync`) than it is to update the configurations on a network full of routers.



A Word About Broadcast Time

- NTP servers can be configured to "broadcast" time info to local clients
- Accuracy is usually poor
- Savings on network traffic is usually not that significant

Avoid using broadcast!

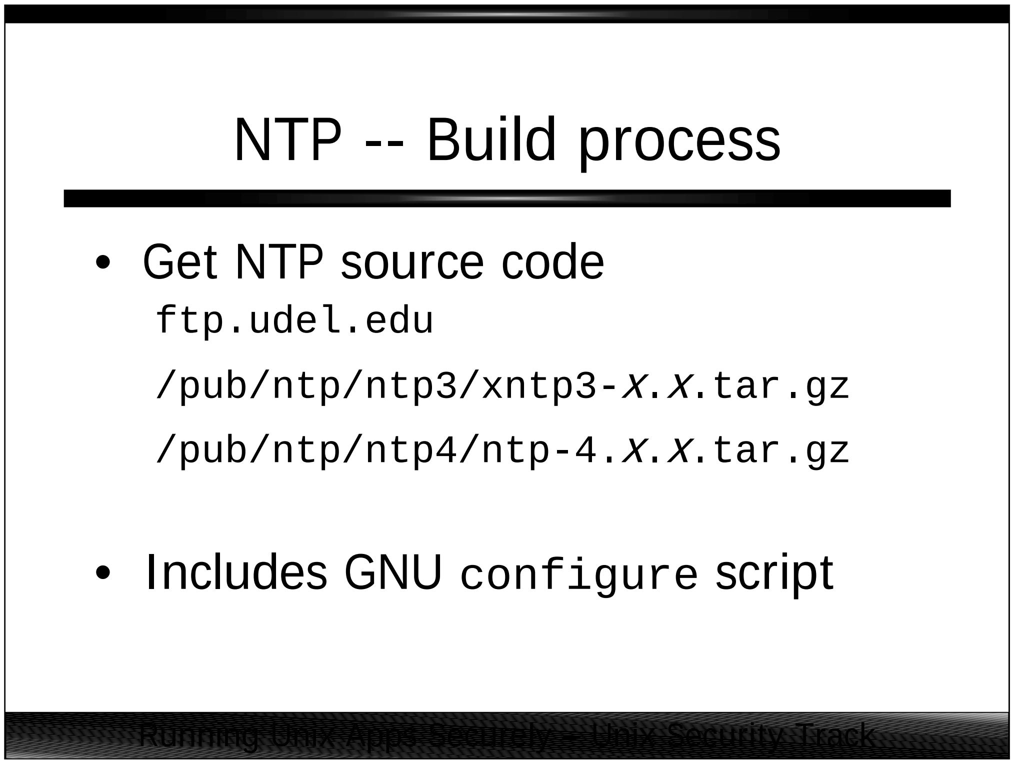
NTP allows servers to broadcast time information to local clients (rather than having the local clients continuously synchronizing against one or more central servers). This can save some network bandwidth, though NTP is generally such a lightweight protocol that the savings is negligible.

The problem is that broadcasts are inherently unreliable– a client may regularly miss the broadcast updates and its clock may being to drift off true. Also, the client may be unable to accurately determine the delay between server and client, again resulting in inaccurate time on your client machines.

On the whole, the savings in network bandwidth (especially on modern switched network fabrics) don't seem to outweigh the disadvantages in using broadcast time synchronization.

Installation and Configuration

Installation and Configuration Examples includes not only actual NTP configuration files but also provides information on obtaining and building the Open Source version of NTP.



NTP -- Build process

- Get NTP source code

  `ftp.udel.edu`

  `/pub/ntp/ntp3/xntp3-`*`X.X`*`.tar.gz`

  `/pub/ntp/ntp4/ntp-4.`*`X.X`*`.tar.gz`

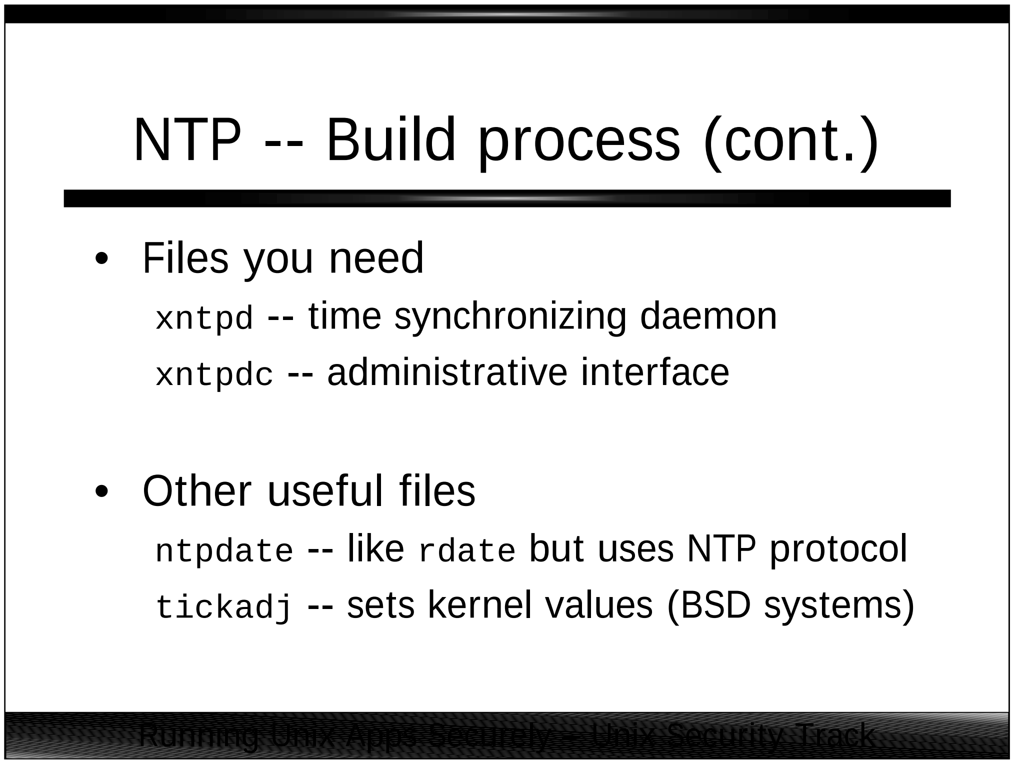
- Includes GNU `configure` script

Note that it may be unnecessary to build NTP from source– most vendor-supplied operating systems now include NTP v3 with the base OS.

The build process for the Open Source NTP distribution is extremely easy thanks to a GNU `configure` script and significant effort on the part of the NTP porting community. Once the configure script is completed, the administrator need only run `make` and then `make install`.

By default NTP will install itself under `/usr/local`. The administrator may change this path by specifying `--prefix` on the `configure` command line:

```
configure --prefix=/opt/ntp
```



NTP -- Build process (cont.)

- Files you need

  `xntpd` -- time synchronizing daemon

  `xntpdc` -- administrative interface

- Other useful files

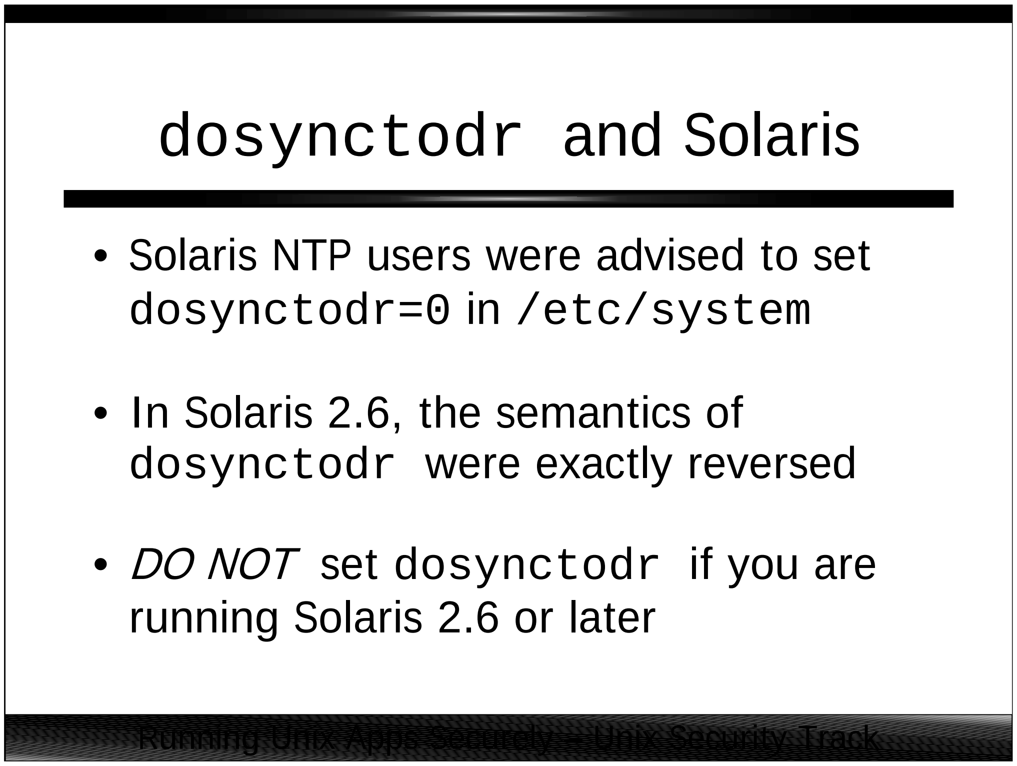
  `ntpdate` -- like `rdate` but uses NTP protocol

  `tickadj` -- sets kernel values (BSD systems)

Running Unix Apps Securely -- Unix Security Track

The `xntpdc` program allows the administrator to interrogate running NTP servers for statistical information and current values such as how much variance there is between the local server and the hosts it is synchronizing against. The `xntpdc` program allows the administrator to query NTP servers on other hosts, including hosts at other organizations! Try the command `xntpdc –p <host>` to get information about which machines a given `<host>` is synchronizing with.

`tickadj` is used to modify various kernel parameters which control features like the granularity of the system clock. This program is not required for SYSV based systems.



`dosynctodr` and Solaris

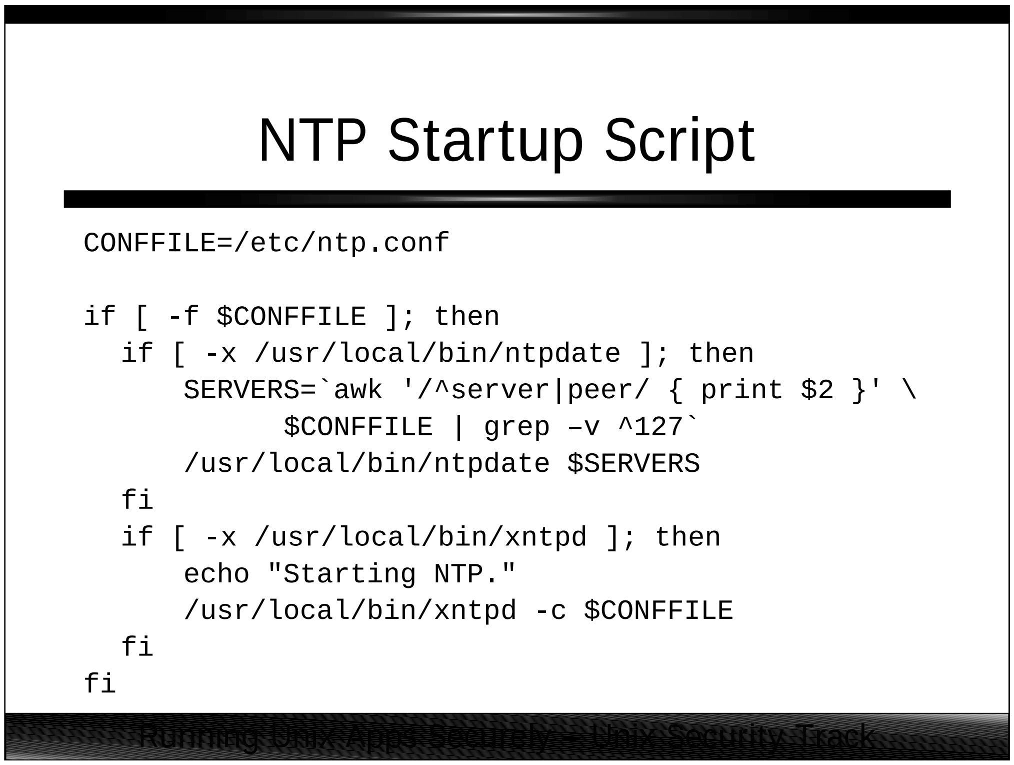
- Solaris NTP users were advised to set `dosynctodr=0` in `/etc/system`
- In Solaris 2.6, the semantics of `dosynctodr` were exactly reversed
- *DO NOT* set `dosynctodr` if you are running Solaris 2.6 or later

Running Unix Apps Securely – Unix Security Track

One of the kernel parameters set by the `tickadj` program on BSD systems is the value of `dosynctodr`– this parameter controls the relationship between the system hardware clock and the system software clock. This parameter sometimes needs to be tweaked on SYSV-based systems as well, but there has been some historical confusion about whether this is necessary under Solaris.

From Sun's "Symptoms and Resolutions Database", SRDB #19195:

> The common lore for setting up `xntpd` on Solaris using the freeware version included the warning to set the kernel variable `dosynctodr` to 0 in the `/etc/system` file thus: `set dosynctodr=0`
>
> When using NTP on Solaris 2.6 or later, the kernel variable MUST be left at the default value of 1. Prior to 2.6 this variable controlled whether or not to rein in the soft clock using the hardware clock, with the result that NTP and the hardware clock would fight for control of the soft clock; thus before 2.6 you had to set `dosynctodr` to 0. At 2.6, every system call that adjusts the soft clock also sets the hard clock, thus while NTP controls the soft clock, the hard clock is also controlled. Setting `dosynctodr` to 0 reverts the behavior back to the pre 2.6 default behavior, having exactly the opposite effect as that intended.
>
> Do not set `dosynctodr` to 0.



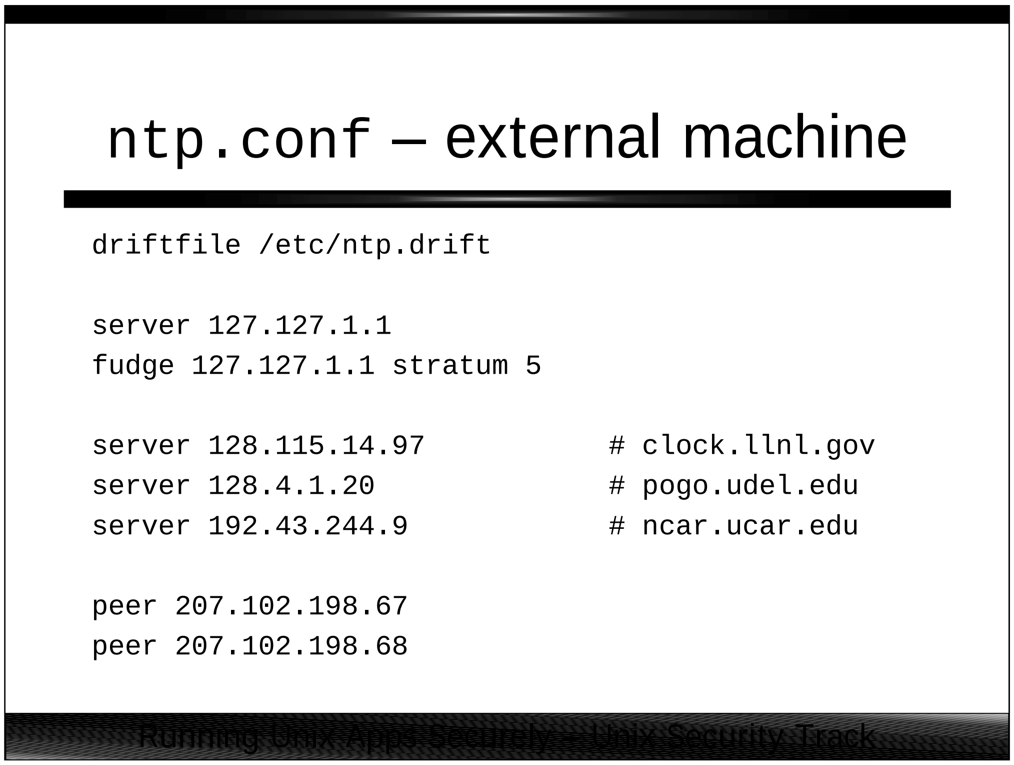
NTP Startup Script

```
CONFFILE=/etc/ntp.conf

if [ -f $CONFFILE ]; then
  if [ -x /usr/local/bin/ntpdate ]; then
      SERVERS=`awk '/^server|peer/ { print $2 }' \
            $CONFFILE | grep –v ^127`
      /usr/local/bin/ntpdate $SERVERS
  fi
  if [ -x /usr/local/bin/xntpd ]; then
      echo "Starting NTP."
      /usr/local/bin/xntpd -c $CONFFILE
  fi
fi
```

Here is a snippet of shell code which can be used to start up the NTP v3 daemon on a Unix machine. Change the pathnames in the script to suit your local installation, and if you're using NTP v4 change `xntpd` to `ntpd`.

Note that the script first calls `ntpdate` before starting the NTP daemon. If your system clock differs too wildly from your external clock sources (for example, after the machine has just been booted for the first time) then your NTP daemon will refuse to synchronize at all. The `ntpdate` call jumps your clock to something approaching the true time and then the NTP daemon will keep your machine chiming away happily.

It turns out that there is an undocumented `–g` flag for `xntpd` which will cause the daemon to synchronize the system clock no matter how far out of synch it may be. You could use this option as an alternative to the `ntpdate` call, but since the flag is undocumented it may disappear in future releases. Thanks to Bob Laughlin (`bel@spawar.navy.mil`) for pointing this option out.



`ntp.conf` – external machine

```
driftfile /etc/ntp.drift

server 127.127.1.1
fudge 127.127.1.1 stratum 5

server 128.115.14.97           # clock.llnl.gov
server 128.4.1.20              # pogo.udel.edu
server 192.43.244.9            # ncar.ucar.edu

peer 207.102.198.67
peer 207.102.198.68
```

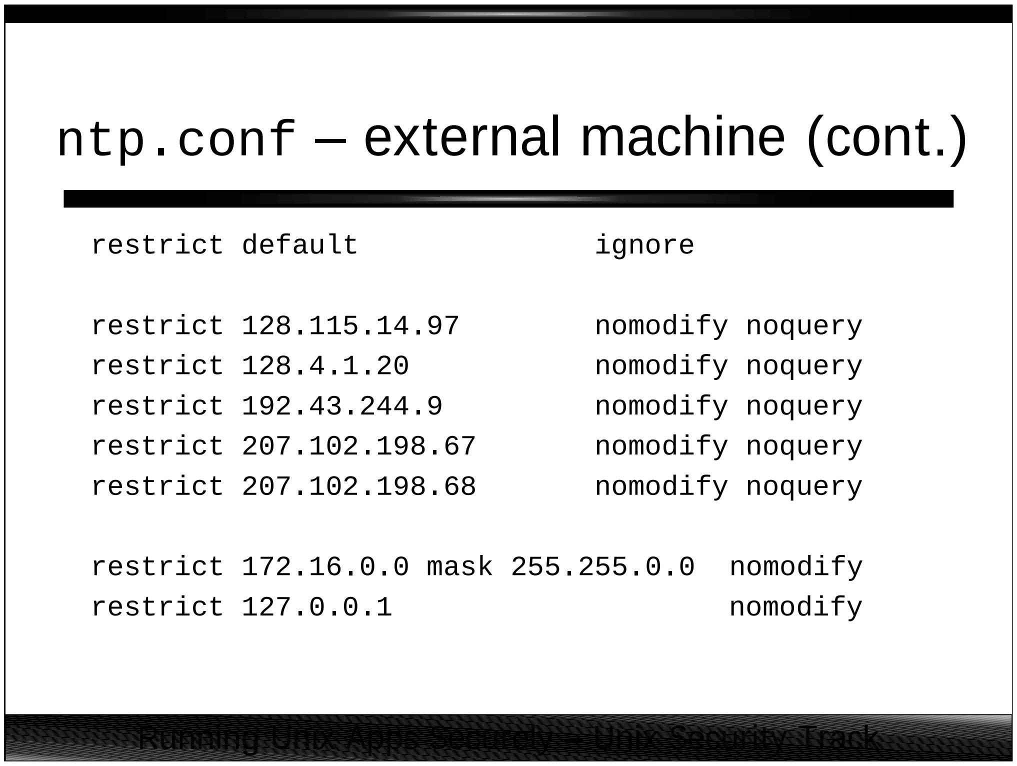
Here is an appropriate configuration file for one of the "Outside Servers" which gets timing information from the external Internet and disseminates it into your internal organization.

The `driftfile` directive specifies the full pathname of the file where NTP should store data on how much the local system clock tends to drift away from accurate network time. Drift information is written to a file so that your machine doesn't have to start re-calculating this quantity from scratch every time your system reboots.

The next two lines define a pseudo-clock at stratum 5– we use a relatively low stratum number since this machine will be at the root of the time hierarchy for your organization. The magic IP address `127.127.1.x` causes NTP to load the pseudo-clock "driver" (other bogus `127.127.x.x` addresses load drivers for other clocks, like GPS receivers and Cesium clocks), and the last octet specifies that this is the first instance of the given driver (theoretically a given machine might have multiple instances of a given clock attached, though this is rare).

Next we peer with three external secondary time servers. Again, please don't be a *clock sucker*– contact the administrator of each external server before writing your configuration file.

The last two lines cause this server to peer with the other two local "Outside Servers".



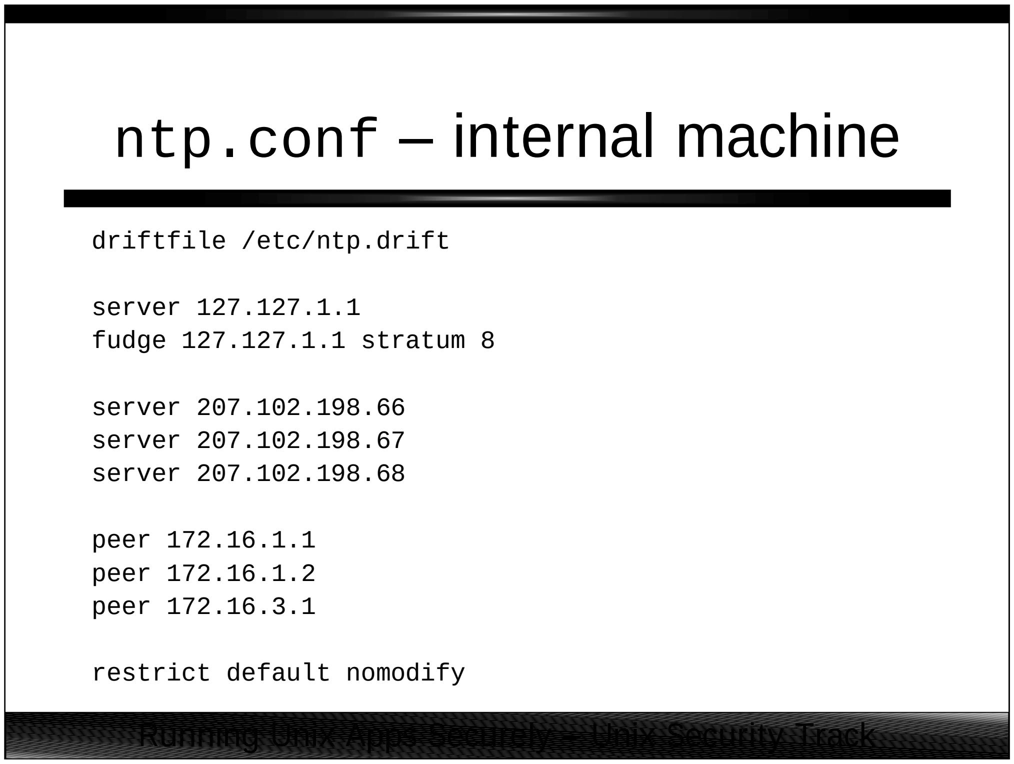
`ntp.conf` – external machine (cont.)

```
restrict default                ignore

restrict 128.115.14.97          nomodify noquery
restrict 128.4.1.20             nomodify noquery
restrict 192.43.244.9           nomodify noquery
restrict 207.102.198.67         nomodify noquery
restrict 207.102.198.68         nomodify noquery

restrict 172.16.0.0 mask 255.255.0.0  nomodify
restrict 127.0.0.1                    nomodify
```

The `ntp.conf` file also allows the administrator to configure which machines get various levels of access to the local server. It is particularly important to configure strong access controls on NTP servers which are accessible from the outside world.

Here we define a default security policy that all NTP protocol messages from all machines are `ignore`d. This means that by default no machine can synchronize time with the local machine and that all administrative requests and requests for information are ignored.

With that default policy, we then configure specific exceptions for hosts we need to communicate with. In particular, we need to be able to synchronize time with our external time sources and peers but those machines should not be able to modify our running configuration, nor should they be able to query the local server for information (in case those external servers are compromised, we don't wish to give away additional information to the attacker). We do want to allow administrators on our internal 172.16.0.0 networks to be able to query the local server (so they can debug problems), but nobody should be able to perform remote reconfiguration. Similarly, we allow and administrator on the local machine to get information but not to perform run-time modifications (since an external attacker could theoretically spoof messages from the 127.0.0.1 address).

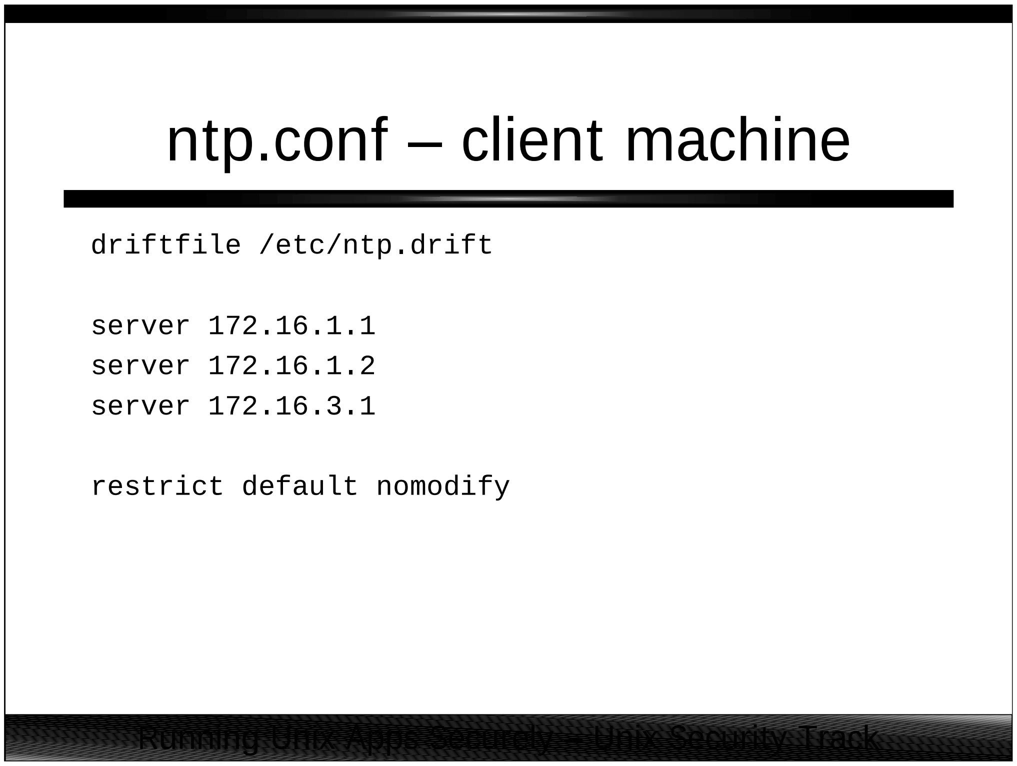


`ntp.conf` – internal machine

```
driftfile /etc/ntp.drift

server 127.127.1.1
fudge 127.127.1.1 stratum 8

server 207.102.198.66
server 207.102.198.67
server 207.102.198.68

peer 172.16.1.1
peer 172.16.1.2
peer 172.16.3.1

restrict default nomodify
```

This is a typical configuration file for one of the internal servers in our example. The host is in a master/slave relationship with the "Outside Servers" and peers with other internal time servers. We configure a pseudo clock at a relatively high stratum value, just in case we lose connectivity to all other internal and external time sources.

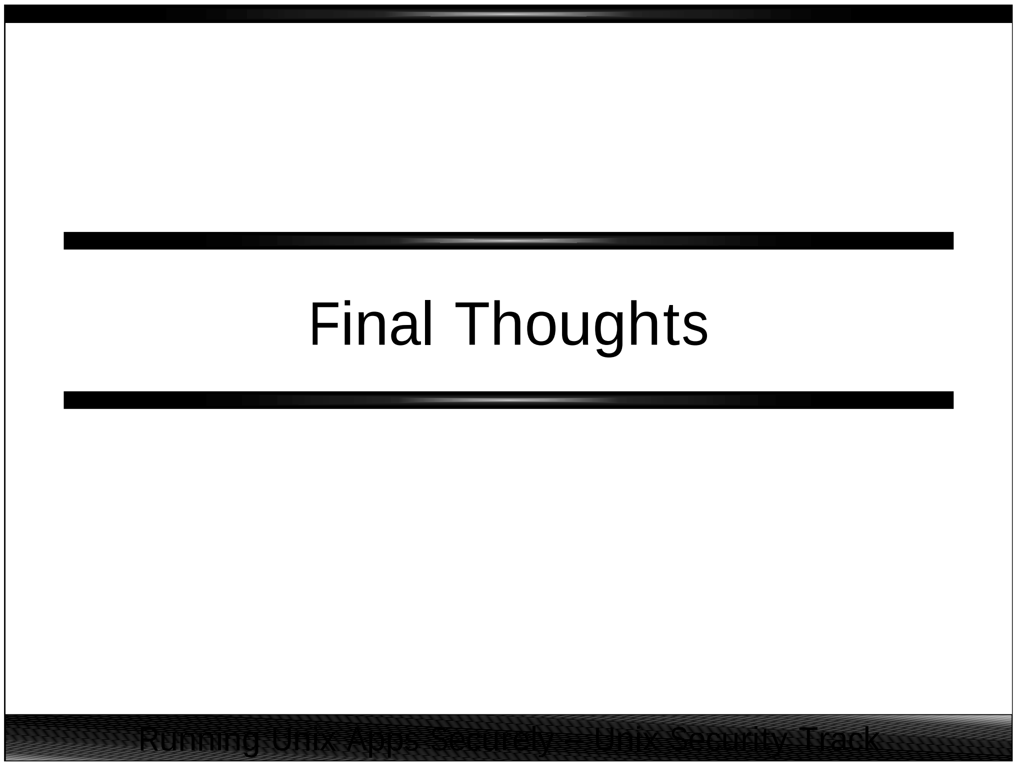
Note that we use a less restrictive default security policy internally. We use this less restrictive configuration assuming that the firewalls for the organization prevent external hosts from injecting NTP traffic onto our internal networks. In a more open network environment, it may be advisable to use a more restrictive (and therefore more complicated) set of access controls.



ntp.conf – client machine

```
driftfile /etc/ntp.drift

server 172.16.1.1
server 172.16.1.2
server 172.16.3.1

restrict default nomodify
```

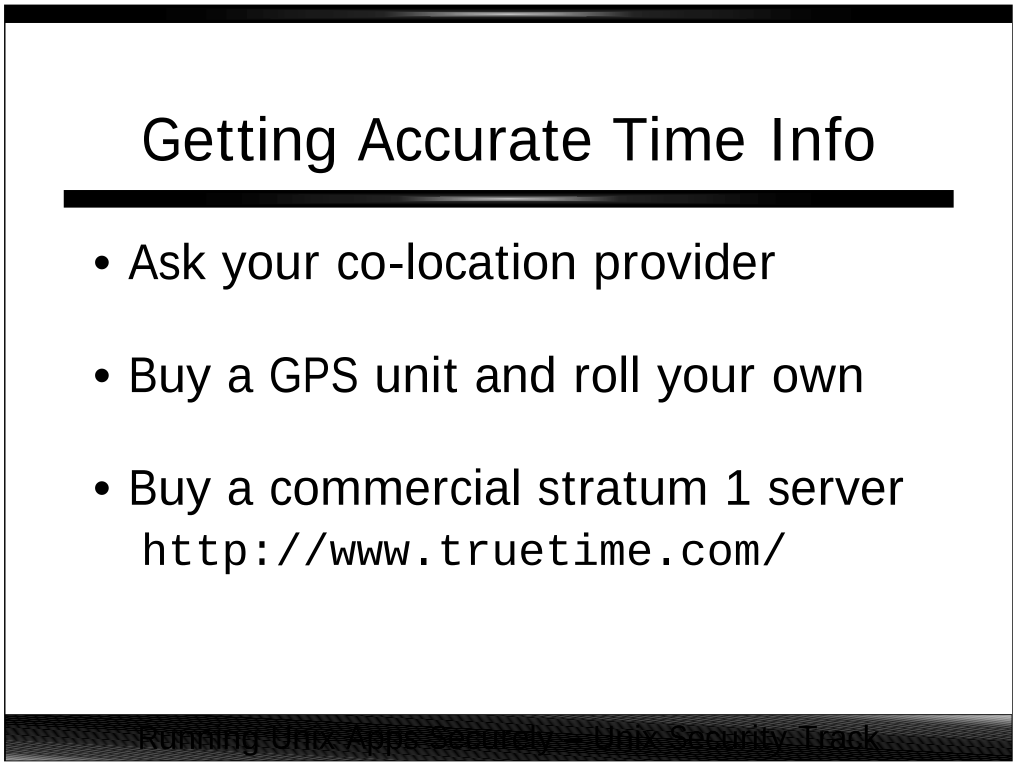
Finally, here's a configuration file that might appear on one of the client machines. The client simply sucks clock from one or more internal time servers. Again we use a fairly unrestrictive access control configuration, assuming that we're protected by our organization's firewalls.



Final Thoughts

Running Unix Apps Securely – Unix Security Track

Accurate, enterprise-wide time synchronization is vital to the security of your organization– both in terms of internal analysis of security data from event logs and other information sources, as well as when you wish to prosecute computer crime cases. Given that NTP is a free and open standard, plus the fact that most modern operating systems already include NTP support, there is no (technical) reason not to enable NTP across your whole organization.



Getting Accurate Time Info

- Ask your co-location provider
- Buy a GPS unit and roll your own
- Buy a commercial stratum 1 server
  `http://www.truetime.com/`

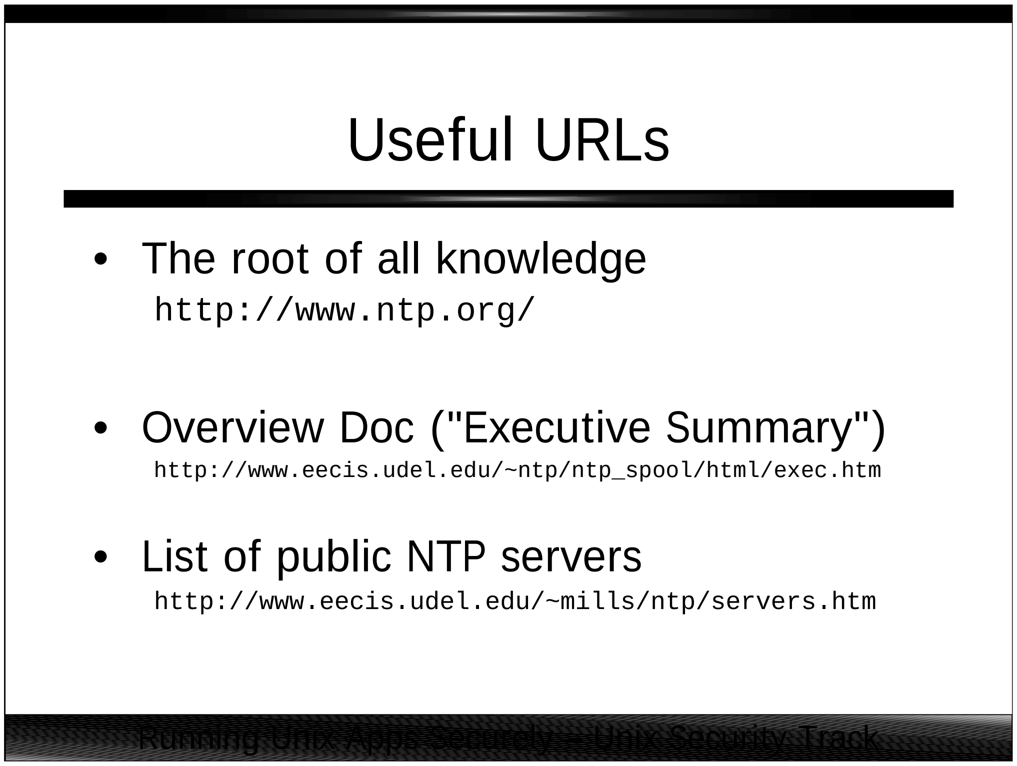
Running Unix Apps Securely -- Unix Security Track

If you are concerned with the accuracy of your organization’s time information (e.g., if you are in a financial environment or make heavy use of Kerberos) you may wish to invest in an accurate clock source of your own.

Hosting and co-location providers often include reliable time servers as part of their service offering, so contact their Network Operations Center or your account rep for more information.

Alternatively, RS-232 compatible GPS clocks cost less than $1000 and drivers for common brands are included in the NTP distribution. A cheap PC running Linux and a little administrative hocus-pocus can give you a stratum 1 server relatively easily.

If you can't afford the time to "roll your own", there are companies which sell fully-integrated stratum 1 servers in the $3000-$4000 price range. Essentially these are "black boxes" which contain a Cesium or GPS clock, an ethernet interface, and enough of an operating system to run an NTP server.



Useful URLs

- The root of all knowledge
  `http://www.ntp.org/`
- Overview Doc ("Executive Summary")
  `http://www.eecis.udel.edu/~ntp/ntp_spool/html/exec.htm`
- List of public NTP servers
  `http://www.eecis.udel.edu/~mills/ntp/servers.htm`

Running Unix Apps Securely -- Unix Security Track

The `NTP.org` server contains links to many, many useful documents. In particular, Dave Mills' "Executive Summary" doc is a good overview of the issues surrounding networked time synchronization. Also available on the `NTP.org` site is a list of publicly accessible stratum 2 servers (with administrative contact information).