NTP Security

Matt Willman
GSEC version 1.4, Option 1
August 2002

Introduction

NTP (Network Time Protocol) is the dominant method for providing accurate time references to local and remote clients. Client software is available for hardware that ranges from embedded solutions to super computers. NTP is important due to the fact that accurate time is critical to almost every component in the global computing infrastructure. Although the parts that make up the concept of accurate time can be debated, keeping accurate time is critical to the on-going operation of our infrastructure.

Little attention has been paid to the security of NTP configurations in the past. Older versions of NTP only had very weak security mechanisms available to them, worsening the situation. Much attention has been focused on the fact that systems need to have accurate time. Little attention has traditionally been paid to the mechanisms that provide that time. This has changed lately with the release of NTP v4 and its attendant security enhancements. The accelerating trend of widely publicized security problems has also raised security awareness for the foundation components of our critical infrastructure.

Providing a trusted source of accurate time reference is more important than ever as our systems and networks continue to grow more inter-connected. While this inter-dependence makes the network as a whole more resilient, it also makes it dependent on the other nodes. The same qualities that comprise our strengths also define our weaknesses. This paper will attempt to cover the basic notions of NTP, configuration suggestions for using NTP in different situations, and the possible problems with poorly secured or improperly configured NTP clients.

How does it work?

The purpose of NTP is to enable the local clock to synchronize itself as close to UTC as possible. This is done by comparing time references (preferably several) against the local clock, factoring in all external influences, then using that calculated value to “discipline” the local clock to bring it closer to UTC. Accuracy is dependent on many external factors, but jitter on a well-behaved LAN approaches milliseconds, while clients across a well-behaved WAN can come within tens of milliseconds of UTC.

NTP uses a series of algorithms for different situations to determine what the most accurate available time source is. These algorithms accommodate network latency, system hiccups, ‘insane’ clocks (false tickers), and many other situations that would cause the local clock or the time reference to be inaccurate. These algorithms also determine the frequency and size of the ‘steps’ used to synchronize the local clock. Steps are the term given to the units of time the local clock is instructed to add or subtract from its local timer.

NTP uses the concept of ‘stratums’ to delineate which systems are closest to accurate time references. There are seventeen stratums, sixteen of which are generally used. The lower the stratum, the more accurate the time source is assumed to be. Stratum zero is generally used for atomic clocks or other highly accurate time sources. You won’t see any network traffic from a stratum 0 server, unless that sever is mis-configured. Stratum one servers synchronize to these sources, and in turn allow tens of clients (stratum two) to synch to them. The stratum two servers in turn let hundreds, (or thousands) of clients use them as a time reference, and so on. The highest stratum, sixteen, is assumed by machines which are not synchronized to anything but their local clock. These systems will refuse to provide time references to anyone until they can find other servers to synchronize from.

This leads to a distributed model, similar to the way DNS functions In NTP there isn’t generally one ‘master’ who serves as the ultimate reference. Rather, in better-configured NTP subnets, there are multiple systems serving as ‘masters’ for accurate time references. All machines that run NTP are clients in one form or another. For example, a workstation that depends on a remote server for time references is a client of that server. This server might be a client to another server that is closer to a more accurate time source. This chain continues to the lowest stratum.

There are many ways to configure machines to act as clients or servers in the NTP hierarchy. More detailed information about suggested configurations for different NTP situations is provided later in this document. If you would like an extremely detailed description of exactly how the NTP protocol internals work, I urge you to reference the excellent presentation written by David Mills[1].

Why does anyone use it?

Accurate time is critical to many of the pieces in all computing infrastructure. NTP provides a way for clients to actively synchronize to UTC from an accurate time source cheaply and effectively. Built into the protocol are safe guards against external tampering, whether malicious or accidental. Newer versions of the NTP protocol include enhancements like autonomous configuration and PKI data encryption. NTP is scalable, cost effective, and has proven itself to be a viable solution for time synchronization needs.

Cryptography, one of the most important parts of any security infrastructure, is dependent on accurate time stamps for multiple facets of its operation. Other pieces of security infrastructure also rely on accurate time; Kerberos, a popular authentication protocol, is totally dependent having accurate local time for correct operation.

> The ticket granting exchange of the Kerberos protocol allows a user to obtain tickets and encryption keys using such short-lived credentials, without re-entry of the user's password. When the user first logs in, an authentication request is issued and a ticket and session key for the ticket granting service is returned by the authentication server. This ticket, called a *ticket granting ticket*, has a relatively short life (typically on the order of 8 hours). The response is decrypted, the ticket and session key saved, and the user's password forgotten[2].

You can see how an inaccurate local clock on the Kerberos server checking the validity of a ticket could have disastrous consequences.

Computer forensics is another high profile area where accurate time is critical. Attacks on computing resources often require no physical interaction with the target system. With little to no physical evidence, the only clues to the intentions or actions of the perpetrator are left behind on the target system after the act has been committed. Knowing the order in which the steps of the attack occurred is critical to determining how the system(s) were compromised. Changes to critical system files, addition of root kits and similar software, verification of log files, all are very difficult to detect without an accurate local clock providing time stamps.

Other pieces of national infrastructure not normally associated with computing infrastructure also rely on accurate time. Machines running NTP, or some subset thereof often provide this accurate time. Security systems are one excellent example. Some of the more sophisticated installations restrict access based on the time of day the access is requested. Anyone with the ability to alter the security systems perception of time would have a free hand. Air traffic control systems, ATMs, traffic ticketing, video surveillance systems, the list goes on. One thing they all have in common is how important accurate local time is.

Having discussed the importance of accurate time, the next question asks itself. Why use something like NTP to synchronize internal clocks when all but the simplest computers come with internal clocks in the first place? Most system clocks are inaccurate due to factors beyond the system's manufacturers control. The inaccuracy of the clock varies from manufacturer to manufacturer, but very few, if any, could be called accurate without irony.

> Clock errors are due to variations in network delay and latencies in computer hardware and software (jitter), as well as clock oscillator instability (wander)[3].

A heavily utilized machine, poorly written software, ambient temperature changes; all contribute to the inaccuracy of the local clock. Something else is needed to correct and compensate for these problems. Preferably something that uses multiple external sources so that one inaccurate reference will not skew the time of the local source. Locally attached hardware time sources are one possible solution, but can quickly become very expensive for more than the smallest and most critical of installations. Purchasing a cheap external time source can often turn out to be a worse solution than the original problem.

Something is needed to provide accurate time references to the hundreds of thousands of potential clients cheaply, effectively, and accurately. This is handled admirably by NTP. The entire design of the protocol enables the most expensive part of time keeping to be as concentrated or dispersed as desired, thus controlling costs and complexity while allowing the many to benefit from the expenditures of the few. In short, NTP provides a way for widely dispersed clients a way to receive a cheap, accurate, and reliable time reference. This is true even over communication links of widely varying quality. In all but the most extreme cases, NTP can provide a valid time reference for the client to synchronize with.

What's the best way to configure it?

There are many different methods for NTP clients to obtain accurate time references. In a situation where broadcasting or multicasting configurations will not work, the available distributed tier model works very well. The ultimate option remains attaching a local time reference that synchronizes to an immaculate source, i.e. a GPS satellite, government sponsored time broadcast, an atomic clock, etc. This solution is often far more expensive than many budgets can handle, especially if there are a large number of clients. However, judiciously attaching these types of local time sources to primary internal NTP servers can have obvious benefits.

Deciding where your lowest tier NTP servers will get their time reference is the first step that should be taken in designing your NTP infrastructure. The number and type of clients plays a critical role in this decision, as does the applicable budget restraints. While NTP servers do not have to be dedicated to the task, dispersing your critical infrastructure pieces across multiple dedicated servers is almost never a bad idea.

If you own an isolated network that does not have Internet connectivity, or very restricted Internet connectivity, a locally attached time source[4] on each NTP server would be a viable choice. This also applies if your need for accurate time overrides all other concerns, including cost. Three servers, each configured to be peers to each other with locally attached time sources would be my recommendation. These three servers would serve as the primary NTP sources for the rest of your network.

Snippets of sample configuration files for three servers that have IP addresses of 192.168.1.20 (server A), 192.168.1.21 (server B), and 192.168.1.22 (server C) follow.

The first few lines of server A’s configuration file would look something like this.

The following line in the configuration file is for the locally attached reference clock. The first two octets (127.127) indicate that this is a locally attached clock, and the last two octets indicate the clock brand.

```
server 127.127.5.0   # local reference clock
```

The following two lines instruct ntpd (the NTP daemon) that these two servers operate at the same stratum as itself, and it should use them for time references.

```
peer 192.168.1.21
peer 192.168.1.22
```

Server B ntp.conf:

```
server 127.127.6.0 # use a different reference clock
peer 192.168.1.20
peer 192.168.1.22
```

Server C ntp.conf:

```
server 127.127.7.0 # use yet another type of reference clock
peer 192.168.1.20
peer 192.168.1.21
```

There are many additional options that could be added to these configuration files, the examples above only cover the clock reference configuration information.

The other configuration extreme would be a public network with plentiful network capacity, where cost is a primary concern. In that situation, at least two servers configured to get their time references from multiple tier one public NTP servers[5] would be a good choice. They should also peer with each other. In this case, clock reference information for server A would look similar to the example below. Server B’s configuration would look very similar, only with different IP addresses for the public NTP servers.

```
ntp.conf:
server 172.16.1.0    # public NTP server
server 172.16.2.0    # public NTP server
server 172.16.3.0    # public NTP server
peer 192.168.1.21    # peer with the other local server
```

(These are randomly chosen addresses)

For corporate networks, a good mix is a three server setup to serve as primary internal NTP servers. Two servers are configured to only obtain their time references from public NTP servers, with the third having a locally attached clock in addition to referencing several public NTP servers. The servers can be configured to reference each other as peers. In this situation, even if you have an extended network outage, the single server with the locally attached time source should keep the clocks in your network relatively close to accurate time. Having at least two independent sources of time references reduces the single points of failure in your NTP infrastructure.

The above configuration suggestions all assume that the clients you want to provide NTP references for are co-located with NTP servers; and you have very few, if any, remote clients. Distributed data centers or a large number of WAN clients demand a more extensive NTP configuration. Unless your distributed data centers are very small, it makes a great deal of sense to duplicate your NTP server infrastructure at the remote sites. If you have a large number of remote clients, you'll reduce the amount of WAN traffic that would otherwise occur. The total amount of NTP traffic is generally insignificant compared to other protocols, but every little bit helps. Other reasons that might be more important include the fact that this type of configuration will allow you to continue providing NTP service to your clients if something were to happen to your primary data center. Especially in today's environment, the possibility of something like that happening must be considered.

Once you have built a solid foundation for your NTP infrastructure, it's time to consider your client needs. If you have a large number of local clients (personal workstations for example), a simple broadcast configuration would be an excellent choice. In this situation, you could setup a secondary layer of NTP servers in an isolated subnet that received their time references from your primary NTP servers, and in turn provided time references to all of your clients. One of your secondary NTP servers configuration file would look like:

```
ntp.conf:
server 192.168.1.20# server A
server 192.168.1.21# server B
server 192.168.1.22# server C

peer 192.168.2.21    # peer with secondary server E
peer 192.168.2.22   # peer with secondary server F

broadcast 192.168.2.255   # provide broadcast service
```

And the ntp.conf file on all of the clients would look like:

```
ntp.conf:
broadcastclient  yes
```

This prevents the large number of clients from overloading your primary NTP server, while providing a readily scalable infrastructure that can be easily expanded by growing your secondary NTP layer. This also allows you to concentrate your resources on securing and administering your NTP servers, while saddling you with a minimal amount of client maintenance.

On the other hand, if you were only interested in providing NTP services to the other servers in your organization and not the personal workstations, a broadcast configuration would not be the best choice in this situation. Configuring your servers as clients to specific NTP servers would be a better choice. This allows you to control which NTP servers your servers synchronize with, and cuts down on the amount of broadcast traffic present on your LAN. Pointing these clients to specific NTP servers also serves as a (weak) defense against someone installing a poison NTP server.

A sample client configuration would be:

```
ntp.conf:
Server 192.168.1.20          # server A
Server 192.168.1.21          # server B
Server 192.168.1.22 prefer# Server C with reference clock
```

The ‘prefer’ keyword gives more weight to the server with the locally attached reference clock. This is another line of defense against outside network interruptions that could affect your NTP servers.

Many corporations do not fit wholly within one situation or the other. They might have a large number of clients that they need to provide NTP time references to, and a small number of servers that they are also concerned about. The configuration suggestions given above could easily accommodate both situations without having to re-architect your entire NTP solution.

Finally, the suggestions above only touch on the time reference information that is present in the ntp.conf file. Many other configuration options exist, security and logging being the most important. A detailed list of all available configuration options is out of scope for this paper. The ‘xntpd’ man page on most Unix machines will contain a list of the possible configuration options, and Caldera has helpfully put their version online[6].

Security options and configuration pitfalls

NTP comprises a critical part of our computing infrastructure. Many other pieces depend on the proper operation of NTP to provide accurate time references for local clock synchronization. While the compromise or failure of your NTP infrastructure might not have a large impact, the impact to other services could be enormous. Whoever controls your NTP infrastructure controls time itself. Compromising NTP opens the door to more sophisticated attacks, some of which include replay attacks, denial of computing resources, undetectable data tampering, etc.

The level of security you apply to your NTP infrastructure will be uniquely dependent on your network and security needs. Available options cover the whole spectrum, from no security on one end, to PKI encryption and authentication of NTP data traffic on the other. How your NTP infrastructure is configured will also determine what security options you apply to your NTP traffic. As with all security, the more secure you make your NTP infrastructure, the more administration overhead you assume.

A simple network with normal security needs might be best served by not using broadcast clients. By configuring your clients to point directly to your NTP servers, you can reduce the impact of someone installing a poison NTP broadcast server on your network and disabling your real NTP servers by some other means. That type of configuration, combined with the appropriate ‘restrict’ keywords in your NTP configuration file will prevent external sources from tampering with your local clock unless they gain control of your client, or all of your NTP servers. Authentication is by IP address of the server, no encryption at any level is provided.

The next level of security involves symmetric key encryption, first available with NTP v3. You can specify multiple keys in your configuration file for different types of access to the server. Each primary server in your NTP infrastructure can be configured with a different key, or each level of access to your clients and servers could have different key sets associated with them. Using symmetric key encryption, along with the appropriate ‘restrict’ clauses in your configuration files will provide a good level of security with minimal impact to your server’s available CPU cycles. The primary disadvantage with this type of configuration is shared with all other symmetric key encryption schemes. In addition, there is administrative overhead associated with distributing the encryption keys over an out-of-band channel. If one of the clients or servers is compromised, all keys on that machine are useless.

The highest level of security you can apply to your NTP traffic is only available with the latest version of NTP, version 4. This version of NTP allows you to use PKI encryption techniques for both authentication and encryption of the NTP data traffic. As with all other PKI setups, additional server and administrative overhead is assumed. However, the maintainers of NTP have made every effort to minimize the impact of using PKI. The new autonomous configuration options and the addition of the PKI code into NTP make the newest version an excellent deployment choice if you have any need for a highly secure NTP infrastructure. The latest status of the inclusion of PKI code into NTP can be had from the IETF[7].

The security options covered above only apply to NTP data traffic and authentication. The security of the servers and networks that NTP runs over should not be marginalized. The most secure NTP infrastructure in the world would be useless if the servers or networks that comprise that infrastructure are easily compromised. There are many excellent guides available on how to improve the security of your servers and networks. One of the better server security guidelines is available from CERT[8].

Conclusion

NTP continues to play a critical role in our global computing infrastructure. Understanding core NTP concepts, how it's used in the real world, and how to secure it is important for it to be a stable foundation piece for that infrastructure. While the compromise or failure of your NTP infrastructure might not have a large initial impact, it represents a very important stepping-stone too much more sophisticated attacks. Taking the time to evaluate your particular security needs and to deploy appropriate security measures can only strengthen that foundation.

This paper has covered the basic concepts of NTP, some sample configuration suggestions, and an overview of the security options available today. Each section could easily be a paper by itself, so I urge you to research each topic more thoroughly[9].

[1] Mills, David. "NTP Clock Discipline Algorithm." 1 June 2001. http://www.eecis.udel.edu/~mills/database/brief/clock/clock.pdf (August 2002)

[2] Neuman, Clifford B. Ts'o, Theodore. "Kerberos: An Authentication Service for Computer Networks." 1994 http://www.isi.edu/gost/publications/kerberos-neuman-tso.html (August 2002)

[3] Mills, David. "Executive Summary - Computer Network Time Synchronization." (Unknown) http://www.eecis.udel.edu/~ntp/ntp_spool/html/exec.htm (August 2002)

[4] Mills, David. "Reference Clocks." (Unknown) http://www.eecis.udel.edu/~mills/ntp/refclock.htm (August 2002)

[5] Mills, David. “Public NTP Primary (stratum 1) Time Servers.” (Unknown) http://www.eecis.udel.edu/~mills/ntp/clock1.htm (August 2002)

[6] Unknown. “xntpd -- Network Time Protocol daemon.“ 5 November 1999. http://docsrv.caldera.com/cgi-bin/man/man?xntpd+1Mtcp (August 2002)

[7] Unknown. “Secure Network Time Protocol (stime).” 4 June 2002. http://www.ietf.org/html.charters/stime-charter.html (August 2002)

[8] Unknown. “CERT® Security Improvement Modules.” 19 June 2002. http://www.cert.org/security-improvement/ (August 2002)

[9] Mills, David. “Time Synchronization Server.” 4 August 2002. http://www.ntp.org (August 2002)