A Security Analysis of Version 2 of the Network Time Protocol NTP: A Report to the Privacy and Security Research Group

Matt Bishop

Technical Report PCS-TR91-154

A Security Analysis of the NTP Protocol

Matt Bishop

Department of Mathematics and Computer Science
Dartmouth College
Hanover, NH 03755

ABSTRACT

The Network Time Protocol is being used throughout the Internet to provide an accurate time service. This paper examines the security requirements of such a service, analyzes version 2 of the NTP protocol to determine how well it meets these requirements, and suggests improvements where appropriate.

1. Introduction

The goal of a time distribution protocol is to deliver continuous, accurate time synchronized with national standards even when leap seconds occur [10]. Such protocols establish a set of primary time reference sources which are directly synchronized with external sources. These may communicate with secondary servers, which in turn may communicate with other (secondary) servers designed to propagate time to hosts on a subnet; the servers propagate the time either by initiating transmission of time messages or by responding to requests from clients seeking the time.

The goal of a time service is to allow a system to synchronize its clock with those of known, accurate primary time servers. This means synchronizing time (so the clocks agree on the time of day) and synchronizing frequency (so the clocks appear to tick at the same rate). However, the propagation of time messages over a network is hindered by transmission delays, unreliable connections, disparity of methods of clients obtaining the time, and heterogeneousness of computing resources. These factors should not affect the synchronization of the clocks, so a time service must provide accurate time even in the face of large (statistical) delays during propagation, as well as being very redundant, so the loss of a single subnet or transmission path does not prevent other portions of the network from obtaining the correct time. Further, the protocol must be flexible enough to work with a variety of client/server interfaces, including having clients continuously poll for the

---

This report is a preliminary report on issues of security and integrity affecting the network time protocol NTP, version 2, done for the Privacy and Security Research Group (a group under the Internet Research Steering Group of the Internet Activities Board). It has not yet been reviewed by the PSRG and hence should be considered preliminary. However, due to increased interest in the issues raised by this report, it is being made available to the community at large.

This work was supported by grant NAG 2-628 from NASA Ames Research Center to Dartmouth College, and by a Burke Award from Dartmouth College.

time, or obtain it by remote procedure calls, as well as in broadcast, multicast, and point-to-point transmission modes.

In what follows, we shall consider only attacks involving the transmission (or hinderance of transmission) of time messages; we shall assume that the messages leave the source uncorrupted, and once they arrive at the target they will not be altered. We make this assumption for simplicity; first, not knowing the operating systems under which these protocols can run, without this restriction we would have to analyze all operating systems which might run the protocols. Secondly, as access to networks is usually easier to obtain than access to individual hosts, the focus of a *network* time protocol's security should be on the *network*. Third, as no system is completely secure, the analysis of any protocol which did not involve an assumption about the nature of the attack being from a network would be rather vacuous.

Five types of attacks on a time service are possible. An attacker could cause a non-time server to impersonate a time server (*masquerade*), an attacker could modify some (or all) time messages sent by a time server (*modification*), an attacker could resend a time server's time messages (*replay*), an attacker could intercept a time server's time messages and delete them (*denial of service*), and an attacker could delay the time messages by, for example, deliberately flooding the network, thereby introducing large transmission delays (*delay*).

The goal of this report is to examine the security of version 2 of the NTP protocol [7][8] with respect to the five attacks described above, and when vulnerabilities are found we suggest remedies. The next section describes version 2 of the NTP protocol (the current incarnation), and the section after that analyzes the attacks in terms of that protocol. The final section suggests improvements to make the NTP protocol more resistent to attacks.

2. Network Time Protocol Version 2

The Network Time Protocol (or NTP)[1] is a protocol designed to meet the above requirements in a wide-area network. It designates several sites as *primary time servers*; these communicate with *secondary time servers* over *synchronization paths* which are said to connect *peers*. The secondary time servers also communicate with other secondary time servers; in addition, each such node serves clients on a subnet. The *stratum number* is a measure of distance from a primary time server, specifically the number of synchronization paths that must be transversed to get from the primary time server to the secondary time server. Because network failures must not affect the

1. These definitions and descriptions are from [10], §1.2.

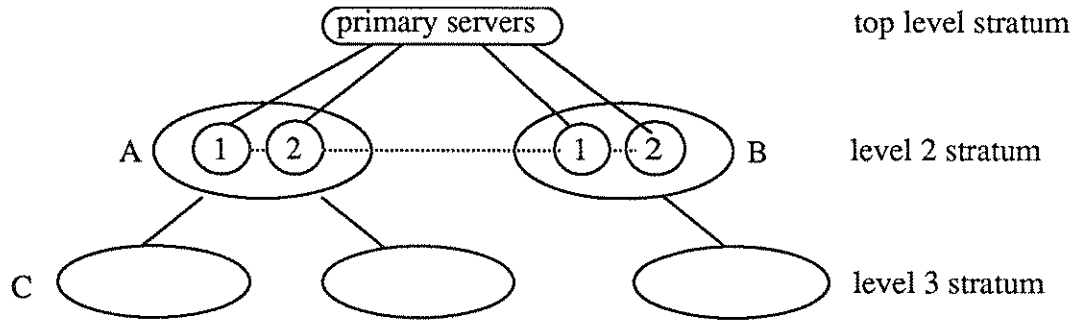


Figure 1. The NTP hierarchy. The ellipses represent sets of cohorts, the solid lines synchronization paths, and the dotted lines paths along which time information is exchanged. A server in C can be synchronized by one in A but not by one in B. If the synchronization path from the primary servers to server 1 of B were to be disrupted, then that server would be synchronized by server 2 in B but not by server 2 in A (as it is in a different cohort). As the shortest path between server 1 in B and the primary servers through its clock source is of length 2, then server 1 of B would be at stratum 3.

availability of the time service, the synchronization paths are not fixed, but may be reconfigured as needed.

Primary time servers are synchronized by an external system (such as radio or atomic clocks) with up to 232-picosecond (2×10^{-10} seconds) resolution[2]. Secondary time servers are synchronized by primary time servers or other secondary time servers with lower stratum numbers. The arrangement is hierarchical, with members of a set (called *cohorts*) initially at stratum level i serving some group at level i+1 (see Figure 1). Members of the group at i+1 may synchronize themselves with any time server in the set, but not with any server not in the set (even if it is at a lower stratum).[3] Membership in a cohort is determined simply by the stratum number of the host and the selection algorithm of the peer that can be synchronized by members of the cohort.

When a message arrives at an NTP time server, it either causes an *association* (instantiation of the protocol machine) to be created, or causes an existing association to act; what happens depends on the mode of the association. The two basic functions are to synchronize another host's clock, or to be synchronized by another host's clock.

2.1. Association Modes[4]

Three operating modes are designed for use on high-speed local area networks, although they may be used on wide area networks as well. The first set allows non-server hosts to synchro-

2. [8], §3.1.2.
3. [8], §2; [13], §3.
4. [7], §3.3; [8], §3.1.1.

nize themselves to NTP servers. An association operating in *client* mode periodically sends messages to its peer; an association operating in *server* mode, which is created when a message from another association operating in client mode arrives, replies with the server's idea of the time, and then terminates; and an association operating in *broadcast* mode sends periodic time messages. The client association may resynchronize the host's local clock, but no association in broadcast or server mode will ever reset its host's time.

The primary and secondary time servers rely on two other modes to synchronize themselves, with higher-level servers synchronizing lower-level ones. An association in *symmetric active* mode periodically broadcasts messages intended to synchronize other hosts. When the messages arrive at these peers, an association in *symmetric passive* mode is created. If the source of the message is at a higher stratum than the current host, a reply is sent and the association terminates. Otherwise, the source synchronizes the current host as indicated by the message; the current host responds with a time message of its own. Normally, the servers at the highest strata will run in symmetric active mode, with servers at lower strata in both symmetric active and passive modes. Note that a host may acquire peers either through receipt of messages or through initialization data read at configuration.

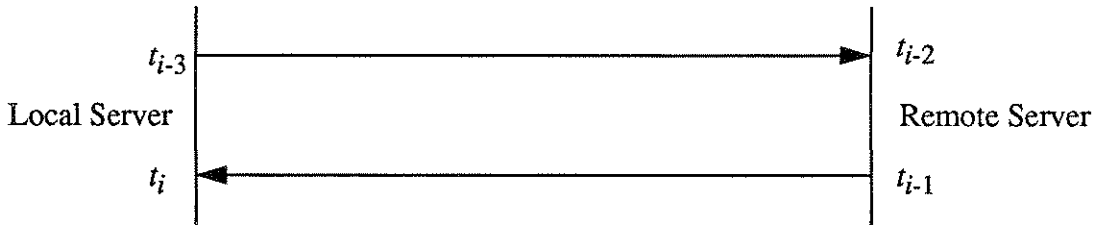
In addition to synchronization messages, NTP allows several types of control messages[5] designed to handle exceptional conditions. These messages do not normally cause synchronization, but instead communicate (or set) information related to the current association, or indicate exceptional events. They are designed for use when no other network management facilities (such as SNMP [1]) are available, and these commands may be sent by other than NTP peers. Further, the NTP specification does not require implementations to be able to process these control messages.

2.2. Selection of Source Peer and Smoothing of Data[6]

NTP uses various algorithms to filter "bad" timestamps from "good," the discriminator including (among other things) how much the newer timestamp is at variance with previous ones.

The first algorithm attempts to improve the accuracy of estimated clock offsets and roundtrip delays by eliminating bad data. From each NTP message the roundtrip delay d and clock offset c are computed (see Figure 2). The values computed from the last eight messages are retained and constitute the *sample*. The first algorithm simply chooses from among these the one with the

5. These messages are described in [7], §9 (Appendix B).
6. The algorithms are described in detail in [7], §4, and are analyzed and evaluated in [8], §6 and [10], §2-§4.



clock offset $c_i = ((t_{i-2} - t_{i-3})+(t_{i-1} - t_i))/2$ and roundtrip delay $d_i = (t_i - t_{i-3})+(t_{i-1} - t_{i-2})$.

Figure 2. Computation of clock offset and roundtrip delay.

lowest delay and uses its associated offset as the estimated clock offset. It also computes an estimate of the sample dispersion based on clock offsets in the sample

The second algorithm uses this estimate to determine which peer should be allowed to synchronize the clock. It first sorts all possible clock sources by stratum number and then by dispersion from the root of the synchronization subnet (that is, up to the primary server synchronizing the node). The list is pruned using various sanity checks and other criteria described in [8]. The elements of this list then are scanned repeatedly, and during each scan the clock dispersion relative to each peer is computed, and that peer with the highest dispersion is eliminated. This repeats until there is only one element in the list; that is the required source.

2.3. Receive and Packet Procedures

Whenever a packet is received, either an error or a packet procedure is called. If the modes of the host and the peer are incompatible (for example, both are symmetric passive), the error procedure is called, the packet discarded, and the association deleted (unless it is preconfigured)[7].

Otherwise, the packet procedure[8] checks that the packet is reasonable and if so, resets internal variables, adjusts the local clock if necessary, and possibly select a new peer to be used as the clock source. It first checks that the packet was not transmitted at the same time as the last one received from that peer (*i.e.*, *pkt.xmt* does not match *peer.org*) to eliminate retransmitted packets, and then checks that the last packet the peer received from the local host was indeed the last one the local host sent to the peer (*i.e.*,*pkt.org* matches *peer.xmt*) to ensure messages are not being received out-of-order. If either condition fails a sanity check is set but processing continues. The association updatesitself to reflect the newly-received packet (see Figure 3 for a list of the internal variables altered) and the polling interval[9]. It then checks that the peer clock is properly synchro-

7. The procedures called for each possible pair of host-peer modes is given in [7], Table 6.
8. This procedure is formally described in [7], §3.4.3.
9. The procedure to do this is described in [7], §3.4.8.

association variable	*set to*	*meaning*
peer.leap	pkt.leap	leap second?
peer.stratum	pkt.stratum	stratum number of peer
peer.ppoll	pkt.ppoll	polling interval
peer.precision	pkt.precision	precision of peer's clock
peer.distance	pkt.distance	estimated delay from primary
peer.dispersion	pkt.dispersion	estimated dispersion from primary
peer.refid	pkt.refid	reference clock identifier
peer.reftime	pkt.reftime	time peer last updated
peer.org	pkt.xmt	when peer sent message
peer.rec	sys.clock	when peer's message received

Figure 3. Association variables set on receive.

nized and authenticated, and that the peer's stratum level is at least as high as that recorded in the packet; if any of these conditions fail, the sanity check is set. Next, the association validates that two-way communication with the peer exists (*i.e.*,*pkt.org* and *pkt.rec* are not 0). If the sanity check is set or two-way communication does not exist, the procedure exits. Otherwise, the packet procedure estimates the round-trip delay and clock offset with respect to the peer, and on some local area networks, a correction factor involving the field *peer.precision* may be applied. If appropriate, the clock update procedure[10] is invoked to update the local clock. (Figure 4 summarizes this procedure.)

Once the packet procedure is finished, the receive procedure resumes. If the peer is in client

```
if (time packet transmitted = time last received packet transmitted) then
        sanity := true;
if (time peer received last packet from host <> time last message sent to peer) then
        sanity := true;
(* update association variables in Figure 3 *)
if (peer clock not synchronized) or (peer clock not updated for 1 day) then
        sanity := true;
if (not authenticated correctly) then
        sanity := true;
if (peer not preconfigured) and (packet's stratum > peer's stratum) then
        sanity := true;
if sanity then
        (* discard message and exit *)
if (packet originate timestamp = 0) or (time last message received by peer = 0) then
        (* exit; note sanity flag not set *)
(* compute delay, offset, corrections, update local clock *)
```

Figure 4. Summary of the packet procedure.

10. [7], §3.4.5.

mode, or the local host is in server or broadcast mode, the peer must be sent the local time, so after the frequency of polling by the peer is updated, a reply is transmitted and the receive procedure ends. Otherwise, if the sanity check is set, the error procedure is invoked unless the peer is in symmetric active mode and the local host in symmetric passive mode. If the error procedure is not invoked, a flag indicating the peer can be reached is set. In any case the packet is discarded and the receive procedure terminates.

2.4. Transmit Procedure[11]

Associated with each peer is a peer timer which decrements periodically. When that timer is 0, an NTP message is generated and sent to the peer. The transmit timestamp (*pkt.xmt*) is saved to validate the reply. Next, if the packet procedure has not obtained valid roundtrip delay and clock offset measurements from the peer with in the last two time-out intervals, it updates a sample set of (roundtrip delay, clock offset) pairs with (0,0) to skew the estimated dispersion, and then determines if a new time source should be used. The peer timer is reset to the shorter of the peer-host polling interval (*pkt.ppoll*) and the host-peer polling interval (*peer.ppoll*), but not to less than 64 or more than 1024 seconds.[12] This ensures the polling frequency varies only within a specific interval.

Finally, the host-peer polling interval is decreased if the estimated dispersion is larger than 0.5,[13] or increased if it is smaller than 0.5, to balance the need for low dispersion with that for low NTP-related network traffic.

2.5. Security Mechanisms

For reasons discussed earlier, we shall consider only those issues raised by relying on the information transmitted over the network. Each NTP message (also called *packet*) contains the information shown in Figure 5. Three NTP-provided mechanisms access these for security reasons.

2.5.1. Delay Compensation Mechanisms[14]

The most basic mechanism is one to compensate for network delays; it is used to offset problems from statistical irregularities or problems in network connectivity and congestion which might be maliciously created or natural. The precise function used is detailed in [7]; the aspect rel-

---

11. [7], §3.4.1.
12. These bounds are the values of the configurable constants NTP.MINPOLL and NTP.MAXPOLL, respectively; see [7], Table 5, and §3.4.1, §3.4.8.
13. The constant is the value of the configurable parameter PEER.THRESHOLD; see Table 5 in [7].
14. These are described briefly in §2.2.

variable	*representing ...*	*variables*	*representing ...*
pkt.srcadr	peer's address	pkt.precision	precision of peer's clock
pkt.srcport	peer's port	pkt.distance	estimated delay
pkt.dstadr	local address	pkt.dispersion	estimated dispersion
pkt.dstport	local port	pkt.refid	reference clock id
pkt.leap	leap indicator	pkt.reftime	last clock update
pkt.version	version number	pkt.org	when last msg sent
pkt.pmode	mode of peer's association	pkt.rec	when last msg received
pkt.stratum	pkt.stratum	pkt.xmit	when last msg left peer
pkt.ppoll	polling interval		

Figure5. List of NTP message (packet) fields.

evant to this discussion is that the algorithm calculates both the roundtrip delay and the clock offset relative to the peer, and from these applies a statistical procedure to determine estimates used to update the local clock.

2.5.2. Access Control Mechanism[15]

This feature requires that the set of all hosts be partitioned into three subsets: those that are trusted, those that are friendly, and all others. Trusted hosts are allowed to synchronize the local clock; friendly hosts are sent NTP messages and timestamps as appropriate, but may not change the local clock; and messages from hosts in the third subset are ignored. The set of trusted hosts is either preconfigured (at initialization) or configured based upon a trusted ticket service such as Kerberos [17]. The peer address in the NTP packet (*pkt.srcadr*) is to be used as the address upon which access control is based. The implementation of this feature is not specified, although two are suggested (the first, treating all peers configured in symmetric or client modes as trusted and all others as friendly; the second, masking the internet address and looking up the result and the peer mode in a table to obtain the subset to which the peer belongs). This feature ned not be supported for an implementation to conform to the NTP specification.

2.5.3. Authentication Mechanism[16]

A third feature, allowed but not required by the specification, is integral to the packets, and is designed to provide both origin authentication and packet integrity. A major requirement is that the computation of the integrity check be predictable, since it must be done after timestamping, but

15. [7], §3.5. Note that this is not part of the NTP specification. The given methods are recommended ways of implementing access control in the Internet; if another form is more suitable for the environment in which NTP is being run, that form should be used.
16. The authentication mechanism is described in [7], §10 (Appendix C); key assignment is described in [9].

```
if peer.config = 0 then
        if authenticator in message data then
                peer.authenable := 1
        else
                peer.authenable := 0;
if peer.authenable =1 then begin
        peer.authentic := 0;
        if (authenticator in message data) then begin
                peer.keyid := packet.keyid;
                compute_mac(mac, peer.keyid, packet);
                if peer.keyid <> 0 and mac = packet.check then
                        peer.authentic := 1;
        end;
end;
(* if peer.authenable is 0, authentication is not done;      *)
(* otherwise if peer.authentic is 0, the integrity of the    *)
(* packet's contents are suspect                             *)
```

Figure 6. The authentication routine's checking algorithm.

the timestamping must reflect the time needed to compute the checksum. The authentication mechanism described in [7] meets these requirements, and is intended for use only until more general, network-wide, authentication and integrity facilities become available. It uses a cryptographically-based message integrity check; all algorithms and keys are distributed by a mechanism other than NTP, and the keys and algorithms are referenced within the packet by indices.

When a packet is transmitted in authenticated mode, the entire NTP packet except for the authenticator and additional information is checksummed using the active peer's key (if available) or the default key 0 (if not). Note that if the association is symmetric active, client, or broadcast, the key used is that of the local host, whereas if the association is symmetric passive or server, the key used is that of the remote host (or the default key).

When a packet is received, the authentication routine is invoked. If the message contains no authentication information, the authentication and integrity check fails; further, if the peer is not preconfigured (at initialization), authentication for that peer is disabled. In either case the routine exits. However, if the message does contain authentication information, the index number of the peer's key is reset to that in the packet, and the checksum is recomputed and compared to the transmitted checksum. If the key is not the default one, and the checksums match, the authentication and integrity check succeeds; otherwise, it fails. This is summarized in Figure 6.

The authentication and integrity mechanism in [7] uses the Data Encryption Standard

based DEA-1 algorithm (that is, DES in CBC mode) [2] to compute the checksum. The checksum is 64 bits long, and the key index is 32 bits; these 96 bits are appended to the original NTP packet, and their presence is indicated by the length of the packet. (The choice of algorithm is not part of the NTP specification of the authentication and integrity mechanism.)

We should note that for control messages, if the received message is authenticated, the reply is too. If the received message's checksum is correct, the reply is authenticated using the same key; if not, the reply is authenticated using the default key.

3. Analysis of NTP Security Mechanisms

Version 2 of NTP provides two basic security mechanisms: an access control mechanism and an authentication mechanism. Because they are intended to hinder attacks, we discuss them before turning to the specific attacks that may be launched against an NTP host.

3.1. Access Control Mechanism

The access control mechanism conditions access on the internet address in the source field of the packet. If the attacker can generate or modify an NTP packet, the attacker can choose a source address that allows synchronization of the victim. Hence the access control mechanism is redundant from the point of view of network security; the protection it provides is against a compromised time source, which can simply be denied access (or at least, the ability to synchronize the host).

The problem with the recommended access control mechanism is that, in the absense of an integrity checking mechanism, it relies completely on the unauthenticated source address. As an additional measure, access control can be conditioned on a routing basis; that is, a list of all the intermediate nodes the message passes through is built using the IP record route option [3] (which causes intermediate nodes to insert their address into the IP datagram) and determine access based on that list. This allows an NTP packet from a trusted source to be rejected if it passed through untrusted hosts. Of course, as the added routing information is not checksummed cryptographically, it can be altered in transit without detection, just as the source address can; however, if the final part of the route over which the packet travels is trusted, this mechanism may indicate that the packet entered from an untrusted part of the network. Access control may then be based on that information.

3.2. Authentication Mechanism

First, it should be noted that the authentication mechanism is also an integrity mechanism because it guards against the altering of messages while in transit. The authentication provided is simply that of only two parties (the peer and the host) sharing a common key. In particular, if any *n* hosts have the same key, then it will not be possible to determine which of them sent the message.

Secondly, note that the authenticator is excluded from the integrity checking.[17] This means that the key index associated with the generated cryptographic checksum can be altered without detection. In general, this will cause the checksum to be invalidated and the packet to fail to authenticate correctly; however, if a less robust cryptosystem were used to generate checksums, the authentication mechanism might not prevent undetectable modification of NTP messages.

No key distribution mechanism is defined.

The integrity checking algorithm used is subject to various cryptanalytic attacks which have been discussed in the literature [4] because the check is only 64 bits long; however, as these attacks are probabilistic in nature, changing the keys periodically will defeat them.

Although we are assuming no system is penetrated, it is worth noting that the keys are used on a per-host basis, not a per-path basis, so compromise of one host's key can lead to compromise of all the hosts it synchronizes.

We should note that the use of a default key is *not* a weakness, contrary to what it would seem. In Figure 6, notice that if the checksum is computed using the default key, the flag indicating whether or not the packet is authentic is set to 0 (meaning the packet's integrity or origin are suspect). Hence anything authenticated with the default key will be treated as bogus by the other end.

In what follows we shall assume the authentication mechanism is not compromised.

4. Analysis of NTP with Respect to Attacks

This section describes the attacks that might be launched against an NTP server or client, and how NTP handles them.

4.1. Masquerade

Goal. To persuade a timekeeper that the attacker is a peer authorized to synchronize the timekeeper. Note this includes NTP client processes as well as secondary servers.

---

17. See [7], §10.2.

Attack: Send packets to the victim with source address of the time server to be imitated. As both source *and destination* Internet addresses and ports are matched to find the correct peer, an equivalent attack would simply change the destination address within the NTP message.

Effects: If the host being impersonated is known to the victim and allowed to synchronize the victim, in the absence of access control and authentication, the masquerade may be ignored (but not detected) by the sample processing and selection operations. However, if the attacker alters the timestamps to change the clock offsets and roundtrip delays gradually, those algorithms will provide no protection and the victim's clock will drift from that of the time source.

If the host being impersonated is not known to the victim, and the default is to allow non-preconfigured peers to become the clock source, sending messages in such a way that the victim receives at least 8 messages uninterrupted by any other time source could compromise the time server; since the clock filtering mechanisms use the last 8 messages as the sample upon which outliers are discarded, the attacker needs to ensure it controls the elements of the sample. An even simpler method would be to send messages claiming a very low stratum number, as the selection algorithm would tend to make such a host the synchronization source. For this reason, no non-preconfigured peer should be allowed to become the clock source.

Note that although the timestamps are precise to 80 nanoseconds (and hence it is unlikely the attacker can predict the value of the next time stamp), if the attacker can see the transmitted time of any packet sent from the host to the peer (*pkt.xmt*) and transmit a (bogus) packet to the host *before* the peer does so, the masquerade will not be detected; but if the host replies, and the reply arrives after the true peer sends another message, the true peer's message will be rejected as bogus because the *pkt.org* field in that packet will not match the time the host sent its last message to the peer. In this way, the attacker could successfully spoof the peer.

Countermeasures: The use of authentication would preclude this attack. The use of access control does not; however, it does allow one to "turn off" permission for a suspect server to synchronize the local clock. If access control is used, all non-preconfigured peers should be considered "friendly" at best (using the language of §3.1.)

4.2. NTP Message Modification

Goal. To alter a message from one timekeeper to another to cause the recipient to incorrectly re-synchronize itself, or to disable an active association.

Attack: Alter packets sent to the victim.

Effects: By examining the packet procedure it is clear that several variables related to the association may be changed a packet altered in transit *before* the packet alteration is acted upon (see Figure 4). How would such alteration affect the integrity of the recipient's clock?

First, if any of the sanity checks discussed in §2.2 fail, the packet is discarded, the association deactivated (if the message is from a peer that has not been preconfigured), and the clock is not updated. If all the sanity checks are passed, then the clock may be reset (if the strata numbers are correctly related and any access control mechanism indicates the peer is trusted). If not, no harm is done. If the clock is reset, then the fields that the attacker can alter affecting the new time are *pkt.org*, *pkt.rec*, *pkt.xmt*, and *pkt.precision*. However, for the sanity checks to pass, *pkt.org* must match the time the last packet was transmitted, which is stored in the receiving host, so altering *pkt.org* will cause the packet to be dropped. Hence only *pkt.rec*, *pkt.xmt*, and *pkt.precision* can successfully be altered in an attack to change the local system's time.

Altering *pkt.precision* may result in changes to the roundtrip delay for the packet on systems involving high-speed local area networks. In these cases, a "fudge factor" δ is computed using $c + 2^{pkt.precision}$ where c is a system-dependent constant; this accounts for possible discrepancies between the host and peer clocks[18]. Then δ is added to the roundtrip delay. Clearly, by modifying this field appropriately the roundtrip delay can be made (almost arbitrarily) large. It can also be made quite small by choosing a value so that δ is approximately 0; this has as an advantage that the clock source selection algorithm bases its choice of peer in part upon δ.[19]

We should note that the roundtrip delay and clock offset are used to compute both an adjustment to the frequency with which messages are sent to the peer on the local host's initiative (that is, not in response to a peer's message) and to determine which of the set of possible clock sources should be used as the source. Both these computations involve a statistical (weighted) average of the peer clock offsets as well as estimates of the roundtrip delays and clock offsets. Hence altering *pkt.rec*, *pkt.xmt*, and *pkt.precision* can also affect the choice of clock source and the frequency of initiating contact with other time servers.

If the purpose of the attack is something other than incorrectly altering the victim's local clock, a variety of other mechanisms may be used. We consider these by considering the other fields of the transmitted packet:

pkt.leap As these bits are not used by NTP, modifying them does nothing.

18. [7], p. 26.
19. The detailed description of the algorithm is in [7], §3.4.3 and §4.2.

victim's mode	*altered mode*
symmetric passive	symmetric passive, server, broadcast
client	client
server	symmetric passive, server, broadcast
broadcast	symmetric passive, server, broadcast

Figure 7. Mode combinations deactivating non-preconfigured associations

pkt.version If this is changed to the version number of an earlier version of NTP, the packet will be discarded unless specific exception has been made. This would allow a denial-of-service attack, and possibly other types of attacks if the exceptional actions permitted.

pkt.mode Depending on how the mode is changed and the mode of the victim, this can cause the disconnection of an association (see Figure 7); it cannot change a packet from one that does not cause a clock update into one that does cause a clock update. If the source of the packet has a pre-configured association with the victim, however, the packet is discarded without the association being broken.

pkt.stratum If the (original) value is greater than the victim's stratum number, and the altered value is less, then the altered value will replace the original value in the victim's table of peer associations; this peer then becomes eligible to be added to the list of clock sources. Note that access control mechanisms may prevent this if the peer whose packets are being modified is not trusted.

pkt.ppoll This affects the frequency of the polling of the peer. Associated with each host is a polling interval; this interval is copied into the packet field pkt.ppoll before it is sent. At the other end, the time to initiate a message is reset to $2^{\textit{smaller of peer's polling interval and host's polling interval}}$, unless that is larger or smaller than two preset constants (as described in §2.3.) Hence one can affect the polling interval, but only within specified limits.

pkt.distance Altering this field affects the estimated roundtrip delay (dispersion) that the victim perceives from the primary source and so can effect the choice of clock source as well as the frequency of polling that clock.

pkt.dispersion Altering this field affects the estimated dispersion that the victim perceives from the primary source.

pkt.refid Altering this field affects the time reference source that the victim perceives the primary source to be relying on.

pkt.reftime This is used to detect non-updated peer clocks. If it is over one day different than the *pkt.xmt* field, the packet will be discarded and (if the association is not preconfigured) it will be discontinued. In any case, the state variables associated with the association are updated to those of the packet.

Countermeasures: To prevent message modification from escaping detection, the authentication mechanism must be used. To prevent message modification from affecting the local host time even in the absence of detection is not possible as the distance and dispersion fields can be modified; however, the stratum value should be used only if all sanity checks are passed (this is true for non-preconfigured associations, but not true for preconfigured ones) and access controls indicate the connection is trusted (not simply the host)

4.3. Replay

Goal. To intercept and resend NTP messages from one timekeeper to another to cause the recipient to incorrectly resynchronize itself, or to disable an active association.

Attack: Record messages sent at one time and resend them later.

Effects: First, note that the sanity checks in Figure 4 will detect replay attacks if the victim sends any message to the originator of the packet, because one of the sanity checks compares *peer.xmt* (the time the victim sent a last message to the peer) with *pkt.org* (the time the peer last received a message from the victim). Hence for a replay to be effective, either the packet being replayed must arrive at the victim before it sends the peer anything, or the packet must be altered to contain this time. As [9] points out, the latter essentially implies a passive wiretapping to monitor packets from the victim to the peer (given the resolution of the timestamp and the unlikelihood of the attacker's predicting it exactly).

In the former case, it is not possible to flood the victim in order to force all elements of the sample to reflect the replayed time, because the first step of the packet procedure determines if this packet is the same as the one received previously from the peer. If so, it is discarded. Hence the replayed data will mingle with valid (new) data, and the clock filtering and selection algorithms will cause that data to be ignored if the other packets from the peer are accurate. This suggests one possible attack: record two packets from the peer (being careful that nothing is sent from the victim to the peer in that interval), and then rapidly replay the packets, alternating them. This will make

four sample elements be the first packet and the other four the second. However, as the packets are received, the clock offsets and delays computed from the timestamps will become greater and greater, resulting in the peer's estimated dispersion and delay increasing; the clock selection algorithm will simply drop the peer as a valid source (if there are multiple clock sources, the peer will be an outlier; if the peer is the only source, it will at some point have too large an estimated delay). Hence replay will either have a negligable effect, or isolate the victim (equivalent to a denial-of-service attack).

Other problems exist. If the delay is greater than the polling interval or no other message has been sent from the peer to the target, the sanity checks in the receive procedure will detect the replay; but as noted in the previous section, this can still cause various association parameters to be reset. In particular, if the synchronization paths have been reconfigured so the peer's stratum number has dropped (and hence the target's stratum number has dropped), the peer could become a source.

Otherwise the effects are the same as for message modification.

A major effect of a replay attack will be to reset the recipient's clock backwards; as the message is valid but for an earlier time, if the replay is not caught and the victim resynchronizes its clock to the (replayed) time in the packet, the local time will be reset to an earlier time.

Countermeasures: Decreasing the bounds of the polling interval will decrease the window of vulnerability. As an alternative, change the first sanity check in the packet procedure to reject any message with a transmit timestamp older than the last one received, and create a special resynchronize message to be sent when a clock is changed backwards. Then the window of vulnerability exists only when a resynchronization packet is sent.

If authentication is used, a less suitable alternative is to employ route-based access control as described in §3.1. If a packet comes over an untrusted portion of the network and contains a time that is earlier than the current time, it is rejected as a possible replay attempt. It should be noted that therecorded IP routing information is not cryptographically checksummed, so it is not reliable; but if the final portion of the route is over a trusted portion of the network, then it may be possible to determine that the packet entered that part of the network from an untrusted part.

4.4. Delay

Goal. To delay NTP messages from one timekeeper to another to cause the recipient to incorrectly resynchronize itself, or to disable an active association.

Attack: Artificially increase (by various nefarious means) the roundtrip delay of an association.

Effects: This increases the estimate of delay to the peer; if more than 8 packets are so delayed (so the estimate of the delay is more than 8 seconds), the peer whose packets are being delayed cannot be a source. This may result in the target having no source, resulting in a denial of service attack.

Countermeasures: The only way to prevent this is redundancy of clock sources, which NTP currently provides.

4.5. Denial of Service

Goal. To prevent NTP messages from any one timekeeper from arriving at the target of the attack, thereby preventing the target from obtaining the correct time.

Attack: Prevent packets from clock sources from reaching an NTP host.

Effects: This will force the NTP server to run under its own clock, and possibly get far out of synchronization with the rest of the Internet (see Table 7 in [7]) for a list of standard time sources and their drift from the correct time).

Countermeasures: The only way to prevent this is redundancy of clock sources.

4.6. Combined Attacks

A combination of the above actions can also prove quite effective during an attack on an NTP server, especially any other attack combined with a denial of service or a delay attack; such combinations attempt to eliminate or hinder communications between a server and members of its cohort not under the attacker's control. For example, denying service to a secondary server from all but one source, and delaying packets from that source, can cause the victim to drift. So can replaying an alternating pair of packets to a server with but one source; the server's time will oscillate between the two values, and the server will report incorrect times. Such attacks can be best dealt with by dealing with each of the component attacks separately.

5. Suggested Improvements

There are two ways for security mechanisms in NTP to evolve. The first is external to NTP, the second internal. External mechanisms are provided by the network protocols upon which NTP is built; internal protocols assume no underlying security mechanism and implement all such considerations within the NTP protocol. Currently, the latter is the model used; so let us begin there.

5.1. Recommendations for the Internal Mechanisms

Authentication should always be used,[20] and the computation of the interity checksum should include the key index.[21] To be more effective, keys should be issued on a per-path, not a per-host, basis. This has been noted in [9], in which it is also said that "the complexity of assigning a distinct key to every peer path used by a server would be pretty fierce ...." However, such a key assignment system adds a fire wall in that if the key for one peer path is compromised, no other peer paths are affected. Further, the different keys do not affect the time needed for authentication, but merely the time needed to administrate the key distribution. As key distribution is out of the scope of the NTP protocol, we merely note that a certificate-based mechanism as used in [5] could be used to distribute keys on a per-peer path basis.[22] There would be a considerable lag involved in validating the keys, but as noted in [7], "the nature of NTP is quite tolerant to such disruptions [as inconsistent key information while re-keying is in progress], so no particular provisions are needed to deal with them."[23]

The record route option of IP should be used when available, and access control should be based on the routes recorded. Of course this does not prevent altering the route while the datagram is in transit or at an intermediate node, but it is another detail an attacker will have to worry about.[24]

The peer association variables should be changed only *after* the packet has passed all sanity checks. Otherwise there is a chance the packet is bogus or corrupt, and in either case the information in it is not reliable and should not be used.[25]

The legal values of the field *pkt.precision* should be constrained more tightly than is currently done. As of version 2, this field may assume values between -127 and 127 inclusive; it is unlikely that any clock will have precision as coarse as 2^{127} seconds (roughly 5×10^{31} years) or as fine as 2^{-127} seconds (roughly 6×10^{-39} seconds) in the immediate future. Note that this applies only

20. See §3.2, fourth paragraph, and §3.4 of this report.
21. See §3.2, second paragraph.
22. We also note that the protocol used to distribute keys must not rely on NTP for determining the time during which those keys will be valid, because then if the relevant NTP server were tricked into incorrectly setting its clock, all keys would have invalid periods attached, enabling the attacker to force NTP servers to use current keys (thereby disabling periodic key changes, or the changing of compromised keys). The Kerberos protocol suffers from this problem, as key validity (or the lifetime of the relevant ticket) is determined by a timestamp obtained from the Kerberos server which, presumably, would be set using an NTP server [17]. The certificate-based mechanism mentioned above may have its interval of validity set either automatically or manually, and in any case will be examined by the human responsible for the NTP server when it is issued. Hence it does not suffer from this problem.
23. [7], p. 56.
24. See §3.1, second paragraph of this report.
25. See §3.4 of this report.

to systems involving high-speed LANS; *pkt.precision* is used nowhere else.[26]

Currently, eight data points are sampled to estimate the dispersion of the clock offset and the roundtrip delay. This enables attackers to flood the victim with bogus packets. If the sample size can be increased to require more data points, this danger can be diminished. Unfortunately, more than eight points can diminish the stability of the local clock and so diminish the effectiveness of the algorithm [11]. Perhaps allowing some maximum number of packets per polling interval would have the desired effect without affecting the statistics adversely.[27]

The danger of replay arises from the possibility of a system's clock being set *backwards* by a packet from another host. The best way to prevent this is to require a special packet be sent when the clock is to be moved back, and provide a nonce to ensure the packet cannot be replayed. (Note it is not sufficient to reject any packet with a timestamp no newer than the last one received, because a clock may run fast and need to be set back; it must then propagate its change to those for which it is the source.)

Finally, redundancy must be ensured; in particular, no server should have as its source only one other server. NTP does this to a large extent already, but it is imperative that the sets at the various strata contain more than one element. This will limit the effectiveness of delay and denial-of-service attacks.[28]

5.2. Applicability of External Mechanisms

NTP has attempted to provide its own security, with all the resulting problems of any security system. An alternative is to use a security protocol for the underlying transmission mechanism and ignore security considerations at the higher (NTP) level.

There are two problems with such a design. The first is that none of the major security-oriented protocols allow broadcast, because broadcasting unforgeable, authenticated packets would imply the use of a public-key checksumming scheme, and no such scheme runs quickly enough to be used in that context (the best-studied, RSA, runs at 1150 bits/second on a Sun 3/60 [6]; given that the checksum should be on the order of 512 bits, this would mean that at most only 2 packets could be processed per second).[29] The second is that few such protocols are in widespread use.

---

26. See §3.4, paragraph 3, of this report.
27. See §3.5 of this report.
28. See §3.6-§3.7 of this report.
29. Note that this is not a barrier to NTP, since polling is done no more frequently than on the order of a minute [8], §3.3.

The lack of broadcast is not serious between primary and secondary, or secondary and secondary, servers, as these are not expected to use broadcast mode; however, for a secondary server providing time service to other hosts on a LAN, the broadcast mode is used.[30] One alternative would simply be to eliminate that mode of operation, and require workstations on such a LAN to query the secondary server directly (the address being configured at boot time). A second would be to allow broadcast but require confirmation by the resynchronizing workstation having an NTP association that enters client mode when it uses the broadcast NTP message to reset the local clock.

Unfortunately, the availability of such network-level and transport-level protocols is more serious. The current UDP protocol [12] provide for no security beyond that available with IP. The IP options include two relevant here: security and strict source routing.

Strict source routing forces packets to be routed through specific intermediate hosts. If those hosts and the links connecting them are trusted, then the NTP packets can also be trusted. However, in a wide-area environment, such assurances are rare; and the source route is specified as a set of fields within the IP datagram itself. Those fields have no associated manipulation detection code. Hence if any link is vulnerable to an active wiretapper, the source route can be altered and the packet made to go along any route.

The IP security option [14] is designed for the protection of information falling under the U.S. classification scheme (*i.e.*, Top Secret, Secret, Confidential, and Unclassified) and is not appropriate for use here.

So, at this point we must conclude that IP does not provide sufficient underlying security to enable its use as an external security mechanism even if broadcasting is eliminated or designated "not trustworthy."

Other protocols not currently in widespread use may prove more suitable. For example, the SDNS Security Protocols SP/3 [15] and SP/4 [16] provide integrity and authentication; this would require NTP to detect only replay or delaying attacks. But these are fundamental to NTP's nature (one due to the connectionless protocol used, and the other due to the use of statistical algorithms) and so most likely cannot be prevented by the underlying protocol.

6. Conclusion

The NTP protocol is a useful, well-designed protocol designed to be robust under a variety

30. [7], §3.3, p. 21.

of conditions. Like all other protocols, it has security weaknesses, some of which are inherent in the goals of the protocol and some of which are a result of the limits of the mechanisms used to improve security. In this report we have highlighted specific areas where attacks designed to thwart the goals of NTP are possible, and have suggested improvements where appropriate.

The recommendations made here are made from the security analyst's point of view; whether or not they can be implemented without adversely impacting the goals of the protocol is another matter. It may be necessary to experiment, for example to determine how much increasing the sample size would affect the accuracy of the statistical algorithms used in NTP. Further, there are some attacks against which the only defense is redundancy, and that may not be possible in all circumstances.

Acknowledgments: I would like to thank Dave Mills, the author of NTP, for his extensive help in guiding me through the intricacies of the algorithm, his unfailing good humor, and his frankness about security matters; he also graciously reviewed several drafts of this report. As with so many other papers, this is not simply the work of the author alone; I would like to thank the members of the Privacy and Security Research Group, especially David Balenson, Russ Housley, Steve Kent, John Linn, Dan Nessett, Richard Parker, Ken Rossen, Miles Smid, and Dave Solo, for their helpful discussions of attacks on NTP; Dan Nessett, for his careful critiquing of this document; and also Ralph Merkle, for his comments during the discussion of NTP and security.

References

[1] J. Case, M. Fedor, M. Schoffstall, and C. Davin, *Simple Network Management Protocol (SNMP)*, RFC 1157 (May 1990).

[2] Federal Information Processing Standards Publication 81, *DES Modes of Operation* (Dec. 1980).

[3] Information Sciences Institute, *Internet Protocol*, RFC 791 (Sep. 1981).

[4] B. Kaliski, Jr., R. Rivest, and A. Sherman, "Is the Data Encryption Standard a Group? (Results of Cycling Experiments on DES)," *Journal of Cryptography* **1**(1) pp. 3-36 (1988).

[5] S. Kent and J. Linn, *Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management*, RFC-1114 (Aug. 1989).

[6] D. Laurichesse, *Mise En Œuvre Optimisee du Chiffre RSA*, Technical Report LAAS-90052, Laboratoire d'Automatique et d'Analyse des Systemes (Mar. 1990)

[7] D. Mills, *Network Time Protocol (Version 2) Specification and Implementation*, RFC 1119 (Sep. 1989).

[8] D. Mills, *Internet Time Synchronization: the Network Time Protocol*, RFC 1129 (Oct. 1989).

[9] D. Mills, *personal communication* (Feb. 1989).

[10] D. Mills, "On the Accuracy and Stability of Clocks Synchronized by the Network Time Protocol in the Internet System," *ACM Computer Communications Review* (Jan. 1990).

[11] D. Mills, *personal communication* (July 1990).

[12] J. Postel, *User Datagram Protocol*, RFC 768 (Aug. 1980).

[13] Privacy and Security Research Group, "Meeting Minutes" (Jan. 17-19, 1990)

[14] M. St. Johns, *Draft Revised IP Security Option*, RFC 1038 (Jan. 1988).

[15] SDNS Protocol and Signaling Working Group, *Security Protocol 3 (SP3)*, Revision 1.5, SDN.301 (May 1989).

[16] SDNS Protocol and Signaling Working Group, *Security Protocol 4 (SP4), Revision 1.3, SDN.401 (May 1989).*

[17] J. Steiner, C. Neuman, and J. Schiller, "Kerberos: An Authentication Service for Open Network Systems," USENIX Proceedings, pp. 191-202 (Winter 1988).