Kerberos Security
With Clocks Adrift

Don Davis   Daniel E. Geer, Sc.D.

February 13, 1995

Abstract

We show that the Kerberos Authentication System can relax its requirement for synchronized clocks, with only a minor change which is consistent with the current protocol. Synchronization has been an important limitation of Kerberos; it imposes political costs and technical ones. Further, Kerberos' reliance on synchronization obstructs the secure initialization of clocks at bootstrap. Perhaps most important, this synchronization requirement limits Kerberos' utility in contexts where connectivity is often intermittent. Such environments are becoming more important as mobile computing becomes more common. Mobile hosts are particularly refractory to security measures, but our proposal gracefully extends Kerberos even to mobile users, making it easier to secure the rest of a network that includes mobile hosts. An advantage of our proposal is that we would not change the Kerberos protocol *per se*; a special type of preauthentication exchange can convey just enough replay protection to authenticate the initial ticket and its timestamp to an unsynchronized client, without adding process-state to the system's servers.

I used to be Snow White, but I drifted.

– Mae West

1 Introduction

The Kerberos Authentication System [14] provides password security for large networks. Unlike its principal competitors, KryptoKnight [10] and SESAME, [6] Kerberos requires that all of a network's system clocks must be synchronized. At first glance, this does not seem to be a great burden, at least for UNIX networks, but as Kerberos' influence has grown, synchronization has become a substantial impediment to Kerberos' adoption as a uniform networking standard.

Why has clock-synchronization become more difficult? We find that for three areas of explosive growth in the networking industry, there are good reasons for rejecting clock-synchronization. First, in-house wide-area networks have only recently become common. Corporate wide-area networks usually arise by agglomeration, so interdepartmental rivalries often obstruct centralized host management, including time-synchronization. Such practical political strains were less important in the more monolithic academic and engineering networks that adopted Kerberos early. It's clear, though, that monolithic networks are now passé, so Kerberos will have to accomodate such ordinary social tensions if its success is to continue. Second, online-access providers are bringing a massive surge of decentralized participants into the network industry. Obviously, it's commercially and technically infeasible to force synchronization on in-home network customers, and the exchange of goods and services between counterparties not administratively or organizationally linked virtually guarantees that asynchronous clocks will remain the norm.

Third, the rise of mobile computing is bringing the problem of intermittent connectivity into renewed importance. Here, the problem is mainly technical: a time-synch protocol would burden both the laptop processor and its connection-initiation bandwidth. A social aspect is that increasingly, the intermittent user is a client of more than one organization providing remote access. Mobile users are more vulnerable to security breaches than sequestered networks are, but laptops' intermittency and mobility obstruct the most effective approaches to open systems security:

- Intermittency obstructs time-synchronized cryptographic protocols, as we mentioned above.
- At the same time, challenge-response protocols entail extra messages (see below), and these would impose unacceptable long-haul network delays on mobile users.
- Absent cryptography, the only alternative is firewalls and other protocol filters, but to a firewall, mobile users look like intruders.

This dilemma makes a synchronization waiver for Kerberos even more valuable. Once mobile users can authenticate themselves cryptographically, it becomes possible for a firewall or filter to recognize them, so that the home network can enjoy both styles of protection, instead of being denied both.

2 Why Synchronize, and Why Not?

In this section, we explain synchronization's purpose and alternatives that serve the same purpose. So as to review the history of synchronization's role in Kerberos' development.

Why does Kerberos need time synchronization in the first place? Synchronized clocks enable Kerberized applications to reject replay attacks. In a replay attack, an attacker eavesdrops on users as they present their credentials to servers; later, he resends the intact credentials to impersonate the users. A Kerberos user blocks replay by embedding an encrypted timestamp in each credential, thereby sharply limiting the "replay window." Without synchronization, timestamps from users with slow clocks would be indistinguishable from replays.

The alternatives to timestamping are all variations on "challenge and response." [7] In a challenge-response protocol, the credential recipient prevents replay by challenging each sender to encrypt and return a fresh random number, so as to demonstrate currency. The sender proves his identity by using his private key, or his session key, to encrypt the random number. Challenge-response protocols avoid the complication of synchronizing, but they always use at least one more message than a timestamp protocol, to accomplish the same security goal. Thus, it might seem that Kerberos' designers chose to optimize performance with timestamps and synchronization. As it happens, though, this speed/complexity tradeoff was *not* the reason Kerberos' designers chose a synchronizing protocol.

The 1978 Needham-Schroeder protocol, [11] from which Kerberos descends, used challenge and response to protect authentication credentials from replay. Three years later, Denning and Sacco [5, 2] pointed out that the N-S protocol was particularly vulnerable to compromised session-keys, because its key-distribution tickets made no provision for expiration of keys. They recommended that the tickets be timestamped, so that the session-keys would expire and be renewed regularly. They also recommended replacing challenge-response with timestamps in N-S' session-authentication handshake, and they pointed out that minute-resolution clock-synchronization would suffice to enforce key expirations. Here, synchronization helps to ensure that every connection gets a new session-key. This precaution makes it less profitable to steal session-keys or to attempt their cryptanalysis. Unfortunately, Denning and Sacco did not discuss the importance and difficulty of securing the time-synchronization process itself.

In the mid-80's, MIT's Project Athena incorporated Denning and Sacco's recommendations into their implementation of the Needham-Schroeder protocol, and added other protocols and security features, too. [14] With timestamping in place, N-S became Kerberos' flagship protocol, which we at Athena christened the "Authentication Service." This protocol handles all of Kerberos' password-mediated authentication, principally initial logins and password changes. Kerberos' other protocols enable a user to authenticate to additional services without entering a password anew, and without retaining the password on the local machine, after the user has already logged in.

These newer protocols uniformly use encrypted timestamps to block replay, following Denning and Sacco's recommendation. However, Kerberos was designed to accomodate clock-skews of up to five minutes between clients and servers (though modern time services can synchronize much better than this). Thus, a replayed authentication-message will not be rejected as out-of-date, if it's less than five minutes old (a generous allowance, though not an unreasonable one). To close this security hole, Kerberos introduced a "credentials cache," in which an application server stores each encrypted timestamp it receives for five minutes, the duration of the replay window. Each server should check every new timestamp it receives against its cache, so as to block replays of "fresh" timestamps.

In 1990, 12 years after Needham and Schroeder's paper, and five years after Kerberos' introduction, Bellovin and Merritt of AT&T Bell Labs wrote an important and insightful critique of Kerberos' version 4, which was influential in the design of the current version 5. [1] Along with other problems, Bellovin and Merritt pointed out that Kerberos security depends on *secure* clock-synchronization, and that V4 Kerberos was not itself sufficient to secure a clock-synchronization service. The clearest demonstration of this insufficiency is to consider a computer that is restarting automatically from a power failure, so that its system clock is certainly unreliable. In this situation, the computer cannot be sure of any message's freshness; indeed, if an attacker replays all of a previous day's network traffic, he can mislead the computer into using an old, compromised session-key as if it were fresh, and Kerberos' guarantees evaporate. As Bellovin and Merritt noted, the only way to defeat such an attack is with a challenge-response protocol, which Kerberos currently lacks, and which no current time-service supports.

It turns out that this situation is not merely illustrative, but is actually the crux of the problem. Only when a Kerberos client first comes onto the net, does he need to use a challenge-response handshake to prevent credentials-replay. Application clients and servers meet this circumstance differently, and they must handle synchronization differently, too. Application servers need to use a challenge-response handshake only at bootstrap, to get time-service tickets. Thereafter, a server can trust its system clock, whenever it needs to renew its time-service tickets or other tickets it uses. For application clients, challenge-response is necessary whenever the user logs on to a physically-insecure workstation. Once the challenge-response handshake has assured the client of his initial tickets' freshness, the client does not need to synchronize his clock with the rest of the network. To be able to detect replay, the client only needs to know the difference, or skew, between his clock and the standard clock. [15] Thus, by adding a challenge-response handshake to only the Authentication Service protocol, we can remove Kerberos' circular dependence on a secure time-service.

3 Current Time-Synch Services

Secure NTP is a cryptographically-hardened time service protocol. It relies on each host to maintain a shared key in a disk file, but makes no provision to distribute or refresh these keys. Kerberos can manage NTP's keys, but only under the assumption that the clocks are already synchronized. Secure NTP makes no claim to solve the bootstrap problem. The Open Software Foundation's Distributed Computing Environment (OSF DCE) includes a Secure Time Service, whose security is mediated by the DCE Kerberos server. The DCE time service relies on the host's hardware clock chip to be physically secure, battery-powered, and accurate enough to fulfill Kerberos' secure synchronization needs. DCE explicitly accepts, just as Kerberos always has, that the clocks must be initialized "out-of-band," i.e., by wristwatch. Neither Secure NTP nor DCE's Secure Time Service makes any provision for physically-insecure hosts, which cannot hold long-lived keys on disk, and so cannot participate in either protocol. Our proposal will work well with both of these services, without substantial change to their protocols or software.

4 Proposed Solution

In this section, we describe a "pseudo-preauthentication" protocol for Kerberos, that enables users to get tickets without having synchronized their clocks. We call this "pseudo-preauthentication," because we're abusing a flexible preauthentication extension, specified in Kerberos version 5. [12] Our protocol adheres to the specification, without obstructing true preauthentication. We also describe a mechanism that enables users to present accurate timestamps to Kerberos and to secure applications, without keeping their system clock synchronized. Unlike Bellovin and Merritt's suggested solution for Kerberos' synchronization problems, our proposals add no process-state to the Kerberos server or to the application servers.

True preauthentication proves the user's identity in his initial ticket request, so as to prevent attackers from requesting credentials in the user's name and attempting their decryption with a dictionary of commonly-chosen passwords. Bellovin and Merritt's paper included the first published analysis of Kerberos' vulnerability to this type of attack, and preauthentication is one of the solutions they suggested. There are many possible ways for a Kerberos client to preauthenticate his initial ticket request. [1] In the simplest and least secure way, the login client prompts the user for his password before preparing the ticket request, and uses the password to encrypt a timestamp that authenticates the ticket request to the Kerberos server; the server refuses tickets to clients whose preauthentication fails. Proposals abound to overcome the flaws in this naïve scheme, employing both hardware and exotic software-only protocols, and the Kerberos version 5 specification made flexible provision for vendors to add any and all of these variations to the MIT implementation. [13, 12] To accomodate this variety, the specification document simply allows the client and server to include arbitrary "preauthentication data" in their initial correspondence, and we shamelessly exploit this vagueness in the spec.

For clarity's sake, let's consider first how to initialize a clock securely, on a machine that does intend to synchronize. Suppose Bob is a system server who shares a key K_b with the Kerberos Authentication Server AS, and suppose he is willing to synchronize his clock. Every time he reboots, one of his tasks will be to request

[1] Most Kerberos suppliers have reinforced their Kerberos servers with password-quality controls, which arguably can prevent guessing attacks more definitively than preauthentication can do.

tickets for a secure time service S_t. To do this, Bob will use a challenge-response handshake:

1. $B \longrightarrow AS: \ B, \ S_t, \ N_b$
2. $AS \longrightarrow B: \ \{S_t, \ K_{bt}, \ L, \ T_{bt}\}^{K_b}, \{N_b\}^{K_{bt}}$

Bob's exchange is identical to a usual Kerberos initial ticket request, except that we've conflated a challenge-response handshake into it, formatted as preauthentication data. Note, though, that this handshake does not serve the usual preauthentication purpose of identifying Bob to the Kerberos server AS; instead it proves to Bob that the key K_{bt} and ticket $T_{bt} = \{S_t, \ K_{bt}, \ L\}^{K_t}$ are fresh.

Bob's nonce N_b is a random number, which he can generate from disk-drive randomness [4] or from some other noise source. It is important that Bob's choice for N_b must be immune to external influence; if an attacker can cause Bob to re-issue an old challenge N_{old}, then she can replay correspondingly old tickets $\{S_t, \ K_{old}, \ L_{old}, \ T_{old}\}^{K_b}$, whose session key K_{old} she knows by prior theft. On receiving his time-service tickets from AS, Bob decrypts them with his password K_b, and uses the session key K_{bt} to decrypt the response to his timeliness challenge N_b. When Bob finds that the response is indeed his nonce N_b, encrypted with K_{bt}, he concludes that AS prepared the tickets after receiving N_b. As long as he has never used N_b to request tickets before, this means that the tickets are fresh.

As in the usual Kerberos protocol, AS learns nothing from this exchange about whether it really was Bob who requested tickets, unless other preauthentication data authenticate him. Note that many preauthentication mechanisms, such as smart-card protocols, were originally designed as mutual-authentication schemes in their own right, and do authenticate the server to the client. As long as the preauthentication protocol also protects the initial Kerberos credentials from replay, the client can trust the creation-time to represent Kerberos' current clock-time, without having to use our pseudo-preauthentication handshake.

Now, when Bob uses his new time-service tickets, he sends the usual authenticator, or encrypted timestamp, but it will probably be invalid:

1. $B \longrightarrow S_t: \ T_{bt}, \ \{B, \ \textit{wrong-time}\}^{K_{bt}}$

Note that in practice, Bob will probably put his tickets' creation-time into the authenticator, so *wrong-time* won't actually be far off. However, the timestamp doesn't have to be correct, anyway, because the time service doesn't care about his identity, and it doesn't worry about replayed requests. The time server's response nevertheless concludes the authentication handshake in the standard way, by augmenting the timestamp, and it returns the correct time *t.o.d.* as well:

1. $S_t \longrightarrow B: \ \{\textit{wrong-time}, \text{ “The time is: } t.o.d.\text{”}\}^{K_{bt}}, \ \{B, \textit{wrong-time} + \textit{1}\}^{K_{bt}}$

The first part of the server's response assures Bob that the time-report is fresh, because it includes the timestamp he sent. The second part of the response is somewhat redudant here, but it conforms to the standard Kerberos authentication-handshake.

Suppose now a user Alice wishes to communicate securely with Bob, and suppose that like most users, she prefers not to synchronize. Alice begins her login-session by asking AS for a ticket-granting ticket (TGT), which she'll then use to request tickets for Bob's service:

1. $A \longrightarrow AS: \ A, \ TGS, \ N_a$
2. $AS \longrightarrow A: \ \{TGS, \ K_{a,tgs}, \ L, \ T_{a,tgs}\}^{K_a}, \ \{N_a\}^{K_{a,tgs}}$

This is the same challenge-response handshake that Bob used above, except for the names. On receipt, Alice concludes that her ticket and session-key are fresh, just as Bob did, and she uses the ticket's creation-time to construct a normal TGS ticket-request:

1. $A \longrightarrow TGS: \ B, \ T_{a,tgs}, \ \{A, L_{create}\}^{K_{a,tgs}}$
2. $S \longrightarrow A: \ \{B, \ K_{ab}, \ L, \ T_{ab}\}^{K_{a,tgs}}, \{A, \ L_{create} + 1\}^{K_{a,tgs}}$

As it happens, the authenticators in the TGS protocol don't confer any security benefit; the server doesn't care about replayed requests, and the new ticket's creation-time L_{create} serves as a timestamp to protect the client from replay. [3] Nevertheless, we include the authenticators here as before, so as to minimize our departure from the Kerberos specification.

If Alice, like most users, chooses not to synchronize, she still needs to keep track of the Kerberos server's clock-value, so that she can prepare acceptable credentials, detect replays herself, and anticipate her tickets' expiration. Her ticket's lifetime data L tell her the current value of Kerberos' clock, because L includes the ticket's creation-time L_{create}. Alice can record the skew between her clock and Kerberos', so as to keep track of Kerberos' clock's value. This fixed skew will enable her to prepare acceptable credentials, etc. as usual. [2]

Note that after her initial login with the challenge-response, Alice's other security interactions are perfectly standard, and the rest of the Kerberos protocol is unchanged. However, to support drifting-clock clients, the Kerberos application library would have to be changed to maintain transparently an implicit "session clock" at each end of a Kerberized connection. Each side's skew would be initialized at login or at bootstrap; thereafter, whenever the Kerberos library needs synchronized time, it would add the skew to the local clock. This would allow an application's client and server to use Kerberos for security, even though neither party has synchronized his clock with Kerberos.

Because the Kerberos protocol is unchanged, the drifting-clock clients and synchronized clients would be indistinguishable in their network behavior. Drifting-clock Kerberos clients and servers would fully interoperate with a normal Kerberos installation. If a drifting-clock client requests initial tickets from a Kerberos server that doesn't support challenge and response, the server will reject the preauthentication data. Then, the client can either initialize its session clock on faith, or it can reject the server's tickets as inauthenticable.

5 Conclusion

We have presented a solution to Kerberos' "cold-start" problem in clock synchronization, which provides for secure clock initialization where needed, and for "drifting clock" security where desired. We expect the proposal to gain acceptance rapidly in the broad community of Kerberos' vendors, implementors, and designers, because it requires only minor changes to the Kerberos client library and to the secure time protocols, and because it adds no extra network delays to users' login sequence. Indeed, for Kerberos implementations that already employ preauthentication to protect angainst dictionary attacks, our proposal requires little more than a shift in interpretation, to exploit the fact that with some preauthentication schemes, Kerberos tickets already can be trusted to deliver a secure clock-initialization.

We also expect our proposal, once it's implemented, to greatly improve Kerberos' attractiveness to a variety of commercial network customers and users. Our notion of relaxed, yet secure, synchronization will further lighten administrative burdens and enhance security in large networks. It actually reduces Kerberos' administrative overhead, since most client machines will be able to dispense with time daemons, and it adds neither overhead nor network-latency to secure applications.

[2]Stan Zanarotti, of Dimensional Insight, Inc., devised an unsecured version of this clock-skew trick at MIT, when he implemented MIT's Kerberos clients for the Apple Macintosh. The trick is particularly necessary for the Mac, whose clock is hard to keep synchronized for a variety of reasons. [15]

Intermittency, more than anything else, is the core technical challenge of mobile computing, yet mobile, intermittently connected counterparties have a bigger stake in authenticity than do continuously connected, sequestered network environments. As such, we claim that providing a solution easing Kerberos' synchronized clocks constraint is uniquely valuable because it enables the efficiency and prompt, assured revocation of authority (that is the hallmark of Kerberos authentication) to be broadly applicable to environments that do not and will not have time synchronization services. More broadly, we suggest that as the demands of electronic commerce become better understood, the ability to bridge the boundaries of internally synchronized yet mutually unsynchronized organizations will be shown to have compelling value.

6 Acknowledgments

We thank Barry Jaspan, Marc Horowitz, Jonathan Kamens, John Kohl, Joe Pato, Ted Ts'o, and Stan Zanarotti, for their helpful comments and suggestions. Barry is first, though, because he helped the most.

References

[1] S.M. Bellovin and M. Merritt, "Limitations of the Kerberos Authentication System," *ACM Comp. Comm. Rev.*, **20**(5), pp. 119–132 (October 1990). [research.att.com:dist/internet_security/kerblimit.usenix.ps]

[2] Michael Burrows, Martín Abadi, and Roger Needham, "A Logic of Authentication," *Proc. R. Soc. Lond. A* bf 426(1989) pp. 233-271.

[3] D, Davis and R. Swick, "Workstation Services and Kerberos Authentication at Project Athena," *Technical Memorandum* **TM-424**, MIT Lab. for Comp. Sci. (February 1990).

[4] D. Davis, P.R. Fenstermacher, and R. Ihaka, "Cryptographic Randomness from Air Turbulence in Disk Drives," in *Advances in Cryptology – CRYPTO '94*, Ed. by Yvo G. Desmedt. Springer-Verlag Lecture Notes in Comp. Sci. **839**, pp. 114-120 (1994).

[5] D. Denning and G.M. Sacco, "Timestamps in Key Distribution Protocols," *CACM* **24**(8), pp. 533–536 (August 1981).

[6] T.A. Parker, "A Secure European System for Applications in a Multi-Vendor Environment (The SESAME Project)," *Proc.* 14^{th} *Am. Nat'l. Sec. Conf.* 1991.

[7] Li Gong, "Variations on Message-Replay Detection", 1992. (citation incomplete for now).

[8] Li Gong, "A Security Risk of Depending on Synchronized Clocks", *ACM Op. Sys. Rev.*, **26**(1) pp. 49–53 (1992).

[9] S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer, *Project Athena Technical Plan*, Sec. E.2.1: "Kerberos Authentication and Authorization System," (Cambridge, Mass.) M.I.T. Project Athena internal document, Dec. 21, 1987.

[10] R. Molva, G. Tsudik, E. Van Herreweghen, and S. Zatti, "KryptoKnight Authentication and Key Distribution System." [jerico.usc.edu:pub/gene/kryptoknight.ps.Z]

[11] R. M. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *CACM*, **21**(12), pp. 993-999 (December, 1978).

[12] C. Neuman and J. Kohl, "The Kerberos Network Authentication Service (V5)," Internet RFC 1510, September 1993.

[13] J. Pato, *Using Pre-Authentication to Avoid Password Guessing Attacks*, (Cambridge, Mass.) M.I.T. Project Athena (December 1992).

[14] J.G. Steiner, C.Neuman, and J.I. Schiller, "Kerberos: An Authentication Service for Open Network Systems", *USENIX Winter Conference Proceedings*, February 1988. [athena-dist.mit.edu:pub/kerberos/doc/usenix.PS]

[15] S. Zanarotti, of Dimensional Dynamics, Inc.; personal communication.