Volume 4, Issue 6, June 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Addressing Kerberos 5 Security Issues

Ms. Anjani Gupta*
Assistant Professor-IT, GHEC
India

Mr. Anuj Gupta
M.Tech CSE , Bahra University
India

Abstract*— MIT, Kerberos is a widely-implemented and widely-deployed authentication substrate with a long history in various communities and vendor products. Version 5 of the Kerberos protocol incorporates new features are a step up from traditional security in networked systems; extensions were needed to allow its wider application in environments with different characteristics than that at MIT. From a security perspective, a number of limitations and problems have been introduced with the new versions. As a result the aim of this paper is twofold; firstly to address Kerberos5 security issues and secondly to recommend feasible changes done to a protocol in order to improve the quality of Kerberos.

Keywords- Principal, Realm, Ticket Granting Server, Ticket Granting Ticket, Private key and Session key.

I. INTRODUCTION

Kerberos originated at the Massachusetts Institute of Technology (MIT) in the early 1980s. As computer models began to evolve from central system/dumb terminal to client/server researchers at MIT realized there were a whole new set of issues to resolve. Network users were now able to cause mischief, since they now controlled at least a portion of the computer power. Administrators would need a way to limit and track user actions. An MIT research project was created named Athena. One of the most important issues they found was the passing of passwords across a network in plaintext. The old model had a dedicated line and a single logon there was no threat of attackers listening in. The Kerberos protocol was the result of the work on project Athena.

Before discussing specific problem areas, it is helpful to review Kerberos Version 4. Kerberos 4 was the first officially distributed release. It was made available to the public in 1989. Kerberos 5 is the most current version available, released in 1993. Version 5 has addressed a number of security issues with Kerberos version 4, added new features, and introduced additional encryption methods beyond DES such as triple DES, AES. Kerberos is designed to provide secure symmetric key exchange by leveraging trusted third parties and shared secrets, and it allows mutual authentication of *principals* (clients and services) [1].

II. KERBEROS PROTOCOL

The Kerberos protocol allows a client to repeatedly be authenticated to multiple servers assuming that there is a long-term secret key shared between the client and Kerberos infrastructure. The client long-term secret key was generated using the client's. Exchange between the client and the Kerberos AS (Authentication Server) in Messages 1 and 2 are used only when the user first logs in to the system. Exchange between the client and the Kerberos TGS (Ticket Granting Server) in Messages 3 and 4 are used whenever a user authenticates to a new server. Message 5 is used each time the user authenticates itself to a server[2]. And finally, Message 6 is the mutual-authentication response by the server .A simplified overview of the Kerberos actions is shown in Figure 1.

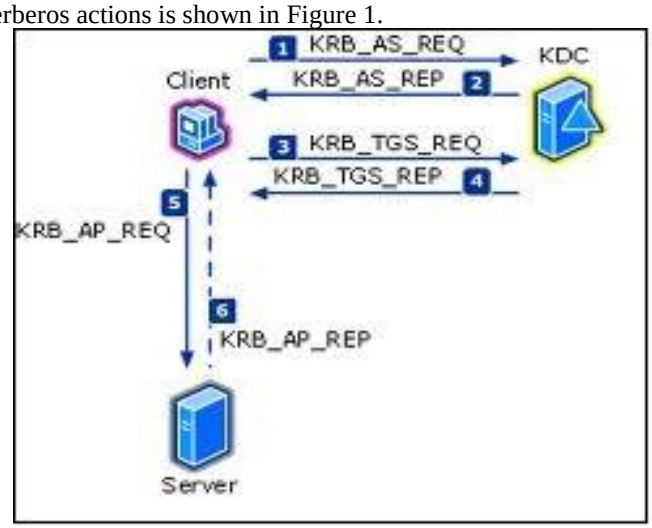

Figure 1. Overview of Kerberos Authentication System

Kerberos is an *authentication* system; it provides evidence of a *principal*'s identity. A principal is generally either a user or a particular service on some machine. A principal consists of the three-tuple

<primaryname, instance, realm >

If the principal is a user — a genuine person — the *primary name* is the login identifier, and the *instance* is either null or represents particular attributes of the user, i.e., root. For a service, the service name is used as the primary name and the machine name is used as the instance, i.e., rlogin.myhost. The *realm* is used to distinguish among different authentication domains; thus, there need not be one giant— and universally trusted — Kerberos database serving an entire company [3].

Table 1: Notations

C	Client principal
S	Server principal
Tgs	Ticket-granting server
K_x	Private key of 'x'
$K_{c,s}$	Session key for 'c' and 's'
$\{info\}K_x$	'info' encrypted in key K_x
$\{T_{c,s}\}K_s$	Encrypted authenticator for 'c' to use 's'
$\{A_c\}K_{c,s}$	Encrypted authenticator for 'c' to use 's'
addr	Client's IP address

Kerberos principals may obtain *tickets* for services from a special server known as the *ticket granting server*, or *TGS*. A ticket contains assorted information identifying the principal, encrypted in the private key of the service.

$$\{T_{c,s}\}K_s = \{s, c, addr, timestamp, lifetime, K_{c,s}\}K_s \ldots (1)$$

Since only Kerberos and the service share the private key *Ks*, the ticket is known to be authentic. The ticket contains a new private session key, $K_{c,s}$, known to the client as well; this key may be used to encrypt transactions during the session. To guard against *replay attacks*, all tickets presented are accompanied by an *authenticator*:

$$\{A_c\}K_{c,s} = \{c, addr, timestamp\}K_{c,s} \ldots (2)$$

This is a brief string encrypted in the session key and containing a timestamp; if the time does not match the current time within the (predetermined) clock skew limits, the request is assumed to be fraudulent. For services where the client needs bidirectional authentication, the server can reply with

$$\{timestamp + 1\}K_{c,s} \ldots (3)$$

This demonstrates that the server was able to read *timestamp* from the authenticator, and hence that it knew $K_{c,s}$; that in turn is only available in the ticket, which is encrypted in the server's private key. Tickets are obtained from the TGS by sending a *request*

$$s, \{T_{c,tgs}\}K_{tgs}, \{Ac\}K_{c,tgs} \ldots (4)$$

In other words, an ordinary ticket/authenticator pair is used; the ticket is known as the *ticket-granting ticket*. The TGS responds with a ticket for server *s*and a copy of *Kc,s*, all encrypted with a private key shared by the TGS and the principal:

$$\{\{T_{c,s}\}K_s, K_{c,s}\}K_{c,tgs} \ldots (5)$$

The session key K_c, *s*is a newly-chosen random key. The key *Kc,tgs* and the ticket-granting ticket itself, are obtained at session-start time. The client sends a message to Kerberos with a principal name; Kerberos responds with

$$\{K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}\}K_c \ldots (6)$$

The client key K_c is derived from a non-invertible transform of the user's typed password. Thus, all privileges depend ultimately on this one key. Note that servers must possess private keys of their own, in order to decrypt tickets. These keys are stored in a secure location on the server's machine [4].

III. SECURITY ISSUES WITH KERBEROS

Although Kerberos was developed to resolve all the issues related to the password-based authentication, many studies reveal that Kerberos is not altogether one hundred percent immune against other form of attacks.

A. Replay Attacks

The Kerberos protocol is not as resistant to penetration as it should be. A number of weaknesses are apparent; the most serious is its use of an authenticator to prevent replay attacks. The authenticator relies on use of a timestamp to guard against reuse. This is problematic for several reasons. The claim is made that no replays are likely within the lifetime of the authenticator (typically five minutes). This is reinforced by the presence of the IP address in both the ticket and the

authenticator. We are not persuaded by this logic. An intruder would not start by capturing a ticket and authenticator, and then develop the software to use them; rather, everything would be in place before the ticket-capture was attempted. Let us consider two examples. Some years ago, Morris described an attack based on the slow increment rate of the initial sequence number counter in some TCP implementations (Morr85). He demonstrated that it was possible, under certain circumstances, to spoof one-half of a pre-authenticated TCP connection without ever seeing any responses from the targeted host. In a Kerberos environment, this attack would still work if accompanied by a stolen live authenticator, but not if a challenge/response protocol was used. Alternatively, an intruder may simply watch for a 'mail-checking' session, wherein a user logs in briefly, reads a few messages, and logs out. A number of valuable tickets would be exposed by such a session, notably the one used to mount the user's home directory. However, for several reasons, we do not think that caching solves the problem. First, on UNIX systems it is difficult for TCP-based (Post81) servers to store authenticators. Servers generally operate by forking a separate process to handle each incoming request. The child processes do not share any memory with the parent process, and thus have no convenient way to inform it —and hence any other child servers — of the value of the authenticator used. There are a number of obvious solutions — pipes, authenticator servers, shared memory segments and the like — but all are awkward, and some even raise authentication questions of their own. To date, we know of no multi-threaded server implementation, which caches authenticators. UDP-based (Post80) query servers can store the authenticators more easily, because a single process generally handles all incoming requests; however, they might have problems with legitimate retransmissions of the client's request if the answer was lost. (UDP does not provide guaranteed delivery; thus, all retransmissions happen from application level, and are visible to the application.) Legitimate requests could be rejected, and a security alarm raised inappropriately. One possible solution, however, would be for the application to generate a new authenticator when retransmitting a request; were it not for the other weaknesses of the authenticator scheme, this would be acceptable [5].

B. Secure Time Services

As noted, authenticators rely on machines' clocks being roughly synchronized. If a host can be misled about the correct time, a stale authenticator can be replayed without any trouble at all. Since some time synchronization protocols are unauthenticated,(Post83, Mill88) and hosts are still using these protocols despite the existence of better ones (Mill89), such attacks are not difficult. The design philosophy of building an authentication service on top of a secure time service is itself questionable. That is, it may not make sense to build an authentication system assuming an already authenticated underlying system. Furthermore, while spoofing an unauthenticated time service may be a difficult programming task, it is not cryptographically difficult. Using time-based protocols in a secure fashion means thinking through all these issues carefully and making the appropriate synchronization an explicit part of the protocol. As Kerberos is proposed for more varied environments, its dependence on a secure time service becomes more problematic and must be stressed. As an alternative, we propose the use of a challenge/response authentication mechanism. As is done today, the client would present a ticket, though without an authenticator. The server would respond with a nonce identifier encrypted with the session key $K_{c,s}$; the client would respond with some function of that identifier, thereby proving that it possesses the session key. Such an implementation is not without its costs, of course. An extra pair of messages must be exchanged each time a ticket is used, which rules out the possibility of authenticated datagram's. More seriously, all servers must then retain state to complete the authentication process. While not a problem for TCP-based servers, this may require substantial modification to UDP-based query servers.

C. Password-Guessing Attacks

A second major class of attack on the Kerberos protocols involves an intruder recording login dialogs in order to mount a password-guessing assault. When a user requests $T_{c,tgs}$(the ticket granting ticket), the answer is returned encrypted with K_c, a key derived by a publicly-known algorithm from the user's password. A guess at the user's password can be confirmed by calculating K_c and using it to decrypt the recorded answer. An intruder who has recorded many such login dialogs has good odds of finding several new passwords; empirically, users do not pick good passwords unless forced to (Morr79, Gram84, Stol88).

To resolve this issue, propose the use of exponential key exchange (Diff76) to provide an additional layer of encryption. Without describing the algorithm in detail, it involves the two parties exchanging numbers that each can use to compute a secret key. The best solution is to support this feature as a domain-specific option. Even exponential key exchange will not prevent all password-guessing attacks. Depending on how carefully the Kerberos logs are analysed, an intruder need not even eavesdrop. Requests for tickets are not themselves encrypted; an attacker could simply request ticket-granting tickets for many different users. An enhancement to the server, to limit the rate of requests from a single source, may be useful. Alternatively, some portion of the initial ticket request may be encrypted with K_c, providing a minimal authentication of the user to Kerberos, such that true eaves dropping would be required to mount this attack. (As we are preparing this manuscript, just such a suggestion is being hotly debated on the Kerberos mailing list. We originally overlooked an alternative avenue for mounting a password-guessing attack. Clients may be treated as services, and tickets to the client, encrypted by K_c, may be obtained by any user. This capability has been suggested as the basis for user-to-user authentication and enhanced mail services (Salt90). However, any such scheme would seem to require repeated re-entry of the user's password, an inconvenience we suspect would not be tolerated. We would prefer to provide the same functionality by having clients register separate instances as services, with truly random keys. Keys could be supplied to the client by the *key store*, described below.)An alternative approach is a protocol described by Lomas, Gong, Saltzer, and Needham(Loma89). They present a dialog with a server that does not expose the user to

password guessing attacks. However, their protocol relies on public-key cryptography, an approach explicitly rejected for Kerberos[8].

D. Spoofing Login

In a workstation environment, it is quite simple for an intruder to replace the login command with a version that records users' passwords before employing them in the Kerberos dialog. Such an attack negates one of Kerberos's primary advantages, that passwords are never transmitted in clear-text over a network. While this problem is not restricted to Kerberos environments, the Kerberos protocol makes it difficult to employ the standard counter measure: one-time passwords. A typical one-time password scheme employs a secret key shared between a server and some device in the user's possession. The server picks a random number and transmits it to the user. Both the server and the user (with the aid of the device) encrypt this number using the secret key; the result is transmitted back to the server. If the two computed values match, the user is assumed to possess the appropriate key. Kerberos makes no provision for such a challenge/response dialog at login time. The server's response to the login request is always encrypted with K_c, a key derived from the user's password. Unless a ''smart card'' is employed that understands the entire Kerberos protocol, this precludes any use of one-time passwords. An alternative (first suggested by T.H. Foregger) requires that the server pick a random number R, and use K_c to encrypt R. This value $\{R\}\ K_c$, rather than K_c, would be used to encrypt the server's response. R would be transmitted in the clear to the user. If a hand-held authenticator was in use, the user would employ it to calculate $\{R\}K_c$; otherwise, the login program would do it automatically. Several objections may be raised to this scheme. First, hand-held authenticators are often thought to be inconvenient. This is true; however, they offer a substantial increase in security in high-threat environments. If they are not used, the cost of our scheme is quite low, simply one extra encryption on each end. A second, more cogent, objection is that if the client's workstation cannot be trusted with a user's password, it cannot be trusted with session keys provided by Kerberos. This is, to some extent, a valid criticism, though we believe that compromise of the login password is much more serious than the capture of a few limited-lifetime session keys. This problem cannot be solved without the use of special-purpose hardware, a subject we shall return to below. Finally, it has been pointed out that a user can always supply a known-clean boot device, or boot via the network. The former we regard a improbable in practice unless removable media are employed; the latter is insecure because the boot protocols are unauthenticated.

E. Inter-Session Chosen Plaintext Attacks

According to the description in the Version 5 draft, (Kohl89) servers using the KRB_PRIV format are susceptible to a *chosen plaintext attack*. (A chosen plaintext attack is one where an attacker may choose all or part of the plaintext and, typically, use the resulting cipher text to attack the cipher. Here we use the cipher text to attack the protocol. Mail and file servers are examples of servers susceptible to such attacks.) Specifically, the encrypted portion of messages of this type has the form:

$$X = (DATA, timestamp + direction, hostaddress, PAD) \ldots (7)$$

Since cipher-block chaining (FIPS81, Davi89) has the property that prefixes of encryptions are encryptions of prefixes, if *DATA* has the form

$$(AUTHENTICATOR, CHECKSUM, REMAINDER) \ldots (8)$$

then a prefix of the encryption of X with the session key is the encryption of(*AUTHENTICATOR*, *CHECKSUM*), and can be used to spoof an entire session with the server. It may be argued that most servers are not susceptible to chosen plaintext attacks. Given that there are easy counters to this attack, it seems foolish to advocate a general format for private servers that does not also protect against it. It should be noted that the simple attack above does not work against Kerberos Version 4, in which the encrypted portion of the KRB_PRIV message is of the form:

$$(length(DATA)\ , DATA, msectime, hostaddress, timestamp + direction, PAD) \ldots (9)$$

as the leading *length* (*DATA*) field disrupts the prefix-based attack[9].

F. Exposure of Session Keys

The term "session key" is a misnomer in the Kerberos protocol. This key is contained in the service ticket and is used in the multiple sessions between the client and server that use that ticket. Thus, it is more properly called a "multi-session key". By making this point explicit, it leads naturally to the suggestion that true session keys be negotiated as part of the Kerberos protocol. This limits the exposure to cryptanalysis (Kahn67, Beke82, Deav85) of the multi session key contained in the ticket, and precludes attacks which substitute messages from one session in another. (The chosen-plaintext attack of the previous section is one such example.)The session key could be generated by the server or could be computed as a session-specific function of the multi-session key.

G. The Scope of Tickets

Kerberos tickets are limited in both time and space. That is, tickets are usable only within the realm of the ticket-granting server, and only for a limited period of time. The first is necessary to the design of Kerberos; the TGS would not have any keys in common with servers in other realms. The latter is a security measure; the longer a ticket is in use, the greater the risk of Key being stolen or compromised. A further restriction on tickets, in Version 4, is that they cannot be forwarded. A user may obtain tickets at login time, and use these to log in to some other host; however, it is not possible to obtain authenticated network services from that host unless a new ticket-granting ticket is obtained. And that in turn would require transmission of a password across the network, in violation of fundamental principles of Kerberos' design.

IV. RECOMMENDED CHANGES TO THE KERBEROS PROTOCOL

Below, there is a list of recommended changes to the Kerberos protocol. Their recommendation is governed by their estimate of the likelihood and consequences of the attack, balanced against the difficulty of implementing the modification.

- A challenge/response protocol should be offered as an optional alternative to time-based authentication.
- Use a standard message encoding, such as ASN .1, which includes identification of the message type within the encrypted data.
- Alter the basic login protocol to allow for handheld authenticators, in which $\{R\}K_c$, for a random R, is used to encrypt the server's reply to the user, in place of the key K_c obtained from the user password. This allows the login procedure to prompt the user with *R*, who obtains$\{R\}K_c$ from the handheld device and returns that value instead of the password itself .
- Mechanisms such as random initial vectors (in place of confounders), block chaining and message authentication codes should be left to a separate encryption layer, whose information hiding requirements are clearly explicated. Specific mechanisms based on DES should be validated and implemented.
- The client/server protocol should be modified so that the multi-session key is used to negotiate a true session key, which is then used to protect the remainder of the session.
- Support for special-purpose hardware should be added, such as the key-store. More importantly, future enhancements to the Kerberos protocol should be designed under the assumption that a host, particularly a multi-user host, may be using encryption and key-storage hardware.
- To protect against trivial password-guessing attacks, the protocol should not distribute tickets for users (encrypted with the password-based key), and the initial exchange should authenticate the user to the Kerberos server.
- Support for optional extensions should be included. In particular, an option to protect against password-guessing attacks via eavesdropping may be a desirable feature.

V. FUTURE WORK

Several issues and limitations are generalized, where synchronization of the clocks has been an important problem. If clocks are not synchronized within a reasonable window Kerberos will report fatal errors and refuse to function. Clients attempting to authenticate from a machine with an inaccurate clock will be failed by the KDC in authentication attempts due to the time difference with the KDC's clock. Time Synchronization will be discussed in next paper.

VI. CONCLUSION

As you can see from the above analysis, the Kerberos protocol has many weaknesses. Some of which stem from the fact that the protocol was specifically designed for used at MIT, and other weaknesses came from poor implementation. To summarize, the security of Kerberos depends critically on synchronized clocks. In essence, the Kerberos protocols involve mutual trust among four parties: the client, server, authentication server and time-server. If any one of these parties fails, then the security of the whole system is compromised.

REFERENCES

[1] S. M. Bellovin and M. Merritt, ‘‘Limitations of the Kerberos Authentication System,’’ Computer Communication Review 20(5), pp. 119-132 (October 1990).

[2] Jennifer G. Steiner, Jeffrey I. Schiller and Clifford Neuman, “Kerberos: An Authentication Service for Open Network Systems” , in M.I.T. Project Athena, Cambridge, Massachusetts (March 30, 1988).

[3] Prof. Hany M. Harb, Prof. Yousef B. Mahdy, Dr. Montaser M. Mohammed and Eng. Yasser F. Uthman , “Overcoming Kerberos Structural Limitations” in IEEE Symposium on Security and Privacy May 14 - 16, 2001. Oakland, California.

[4] John T. Kohl, ‘‘The Use of Encryption in Kerberos for Network Authentication,’’ in Crypto ’89Conference Proceedings, International Association for Cryptologic Research, Santa Barbara, CA(August 1989).

[5] George A. Champine, Daniel E. Geer, and William N. Ruh, ‘‘Project Athena as a Distributed Computer System,’’ IEEE Computer 23(9), pp. 40-50 (September 1990).

[6] Borman, Editor, ‘‘Telnet Authentication: Kerberos Version 5,’’ Internet-Draft, Internet Engineering Task Force, Telnet Working Group (February 1992).

[7] Tom Yu, Sam Hartman, and Ken Raeburn, "The Perils of Unauthenticated Encryption: Kerberos Version 4”. In Proceedings of the Network and Distributed System Security Symposium. The Internet Society, February 2004.

[8] D.L. Mills, "Network Time Protocol," RFC 1119 (September 1989).

[9] W. Diffie and M .E, Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory 6, pp . 644-654 (November 1976).

[10] B. Bryant, Designing An Authentication System: A Dialogue in Four Scenes, Project Athena document, Feb. 1988. (http://web.mit.edu/Kerberos/ di-alogue.html).

[11] G. Bella, and L. Paulson, \Kerberos version IV: In-ductive analysis of the secrecy goals," ESORICS '98, Springer-Verlag, 1998.
[12] S. Sakane et al., \Applying Kerberos to the communication environment for information appliances," IEEE Symposium on Applications and the Internet Workshops (SAINT-w'03), 2003.
[13] T. D. Wu, \A real-world analysis of Kerberos pass-word security," NDSS '99, The Internet Society,1999.
[14] MIT Kerberos Consortium, \The Kerberos web-page," (http://www.Kerberos.org/index.html).
[15] K. Raeburn. Advanced Encryption Standard (AES) Encryption for Kerberos 5, Network Working Group, RFC 3962, 2005. (http://www.ietf.org/rfc/ rfc3962.txt).
[16] Wikipedia, \Kerberos (protocol)," (http://en. wikipedia.org/wiki/Kerberos (protocol)).
[17] T. D. Wu, \A real-world analysis of Kerberos pass-word security," NDSS '99, The Internet Society,1999.
[18] F. Butler, I. Cervesato, A. D. Jaggard, and A. Sce-drov, \A formal analysis of some properties of Ker- beros 5 using MSR," IEEE CSFW '02, pp. 1-16, 2002.
[19] T. Yu et al., \The perils of unauthenticated encryp-tion: Kerberos Version 4," NDSS '04, The Internet Society, 2004.
[20] J. Kohl, and C. Neuman, The Kerberos Network Au-thentication Service (V5), Network Working Group,RFC 1510, Sep. 1993. (http://www.ietf.org/rfc/rfc1510.txt)
[21] http://web.mit.edu/kerberos/
[22] https://help.ubuntu.com/12.10/serverguide/kerberos.html

