INTERNET-DRAFT
draft-ietf-cat-kerberos-pk-recovery-01.txt
Updates: RFC 1510
expires May 23, 1999

Jonathan Trostle
Cisco Systems

Public Key Cryptography for KDC Recovery in Kerberos V5

0. Status Of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as ``work in progress.''

To learn the current status of any Internet-Draft, please check the ``1id-abstracts.txt'' listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

The distribution of this memo is unlimited. It is filed as draft-ietf-cat-kerberos-pk-recovery-01.txt, and expires May 23, 1999. Please send comments to the authors.

1. Abstract

This document defines extensions to the Kerberos protocol specification (RFC 1510, "The Kerberos Network Authentication Service (V5)", September 1993) to enable the recovery of a compromised Kerberos V5 KDC using public key cryptography. The document specifies the recovery protocol which uses preauthentication data fields and error data fields in Kerberos messages to transport recovery data.

2. Motivation

For both secret key based systems and public key based systems, compromise of the security server (KDC in the secret key system and CA or certificate authority in the public key system) leads to a complete breakdown of the authentication service. The difference between the two systems comes when the compromise is detected. Assuming that a root key is intact in the public key system, new high-level certificates can be signed, any suspicious certificates can be revoked, and the system can eventually return to normal

operation without excessive administrator involvement. For a pure secret key based system such as Kerberos V5, the recovery operation is very difficult from an administrative point of view, since all users must receive new passwords out of band.

This document describes an extension to Kerberos V5 that can be used in conjunction with the protocol in [2] (draft-ietf-cat-kerberos-pkinit-07.txt) to allow a KDC to be automatically recovered once the administrator has reinstalled the operating system and loaded (and certified) the new KDC public key. Although the protocols in [2] are a step towards making the KDC recovery problem easier, there are still potentially many secret keys stored on the KDC. For example, when the user private key is stored on the KDC, the user and the KDC share a secret key that is used for authentication. The two main issues for recovery are updating the KDC public key with all clients, which will happen automatically since we assume the KDC public keys are signed as part of a public key infrastructure with a revocation capability, and updating the shared secret keys that are stored on the KDC.

We now describe the requirements for the recovery extension:
(1) Users that use password based keys to authenticate to the KDC (as in section 3.4 of [2] will have those keys automatically changed by the recovery protocol; the users will not have to change their passwords. We will satisfy this requirement by obtaining the secret key K2 of section 3.4 of [2] by hashing the key K1 with a salt value supplied by the KDC. The update operation during recovery consists of changing the salt value. Optionally, the KDC can ask users to change their passwords in order to support recovery in an environment where users use both recovery capable and non-recovery capable clients.
(2) The recovery extension requires the KDC public keys to be signed in certificates as part of a public key infrastructure that includes a revocation capability.
(3) Recovery capable clients must be pkinit [2] capable.

We will use the definitions and ASN.1 structures from [2]; we assume familiarity on the part of the reader.

3. The Recovery Extension Protocol

We now briefly overview the proposed recovery extension. When the recovery operation is launched, the KDC host operating system(s) for the realm along with the KDC database is reloaded from backup media. The new KDC public key certificate is placed into the appropriate certificate database (if needed), and the old certificate is revoked by administrator action. For all principals that have symmetric keys in the database, the keys are zeroized.

To complete recovery, the newly created kdcSalt value (a randomly generated 16 byte string) will be sent to user principals to allow them to update their shared secrets in the KDC database. This exchange

allows users to maintain the same passwords. This task is completed by the following sequence of messages:

```
KDC <-------- AS_REQ message ----------- client

KDC  -------- KRB_ERROR message -------> client
           (error code KDC_ERR_RECOVERY_USER_NEEDED)
            error data: KDC DH parameters, kdcSalt value,
            optional PA-PK-KEY-REP (encrypted user private keys),
            optional change password flag

KDC <-------- AS_REQ message  ---------- client
           (with PA-PK-AS-REQ and PA-PK-RECOVERY-DATA (with new user
            secret key K2 encrypted in Diffie-Hellman shared secret
            key) preauthentication fields)

KDC  -------- AS_REP message  ---------> client
           (with PA-PK-AS-REP preauthentication field)
```

This exchange of messages is only necessary between the KDC and each user principal that has a shared secret key stored in the KDC database.

A recovery capable principal that receives a ticket with an encrypted part using a key with an unexpected kvno, should perform a pkinit [2] AS exchange with the KDC, including the PA-PK-RECOVERY-SET-PRINCKEY-TO-SKEY padata-type to obtain a TGT with a ticket session key that will be used as the new principal secret key. In this case, the KDC would have previously generated the secret key to encrypt the ticket, based on the TGS_REQ from the client, and the database bits indicating that the server principal should have a valid symmetric key but one does not exist in the database. The KDC will always use the symmetric key with the appropriate keytype from the database as the ticket session key when receiving a pkinit request with the PA-PK-RECOVERY-SET-PRINCKEY-TO-SKEY padata-type. The padata-value for this padata-type is an empty octet string.

3.1 Definitions

The proposed extension includes a new algorithm for computing the shared key between a user and the KDC. The new algorithm involves computing the SHA1 hash of a string (kdcSalt) supplied by the KDC concatenated with the RFC 1510 shared key (the key K1 from section 3.4 of [2]) to obtain a new DES key K2 that is shared between the user and the KDC. We propose etype and keytype 16 for this algorithm:

DES-recoverable-key	16

Similarly, we propose the same definition for 3DES where the key

K2 or RFC 1510 shared key is a 3DES key:

3DES-recoverable-key	17

If the KDC expects the client to preauthenticate using the key K2 with a recoverable key keytype, and the client does not preauthenticate, then the e-data for the error KDC_ERR_PREAUTH_REQUIRED will be present containing the kdcSalt value encoded as an OCTET STRING. If the client preauthenticates with the key K2 having keytype DES-recoverable-key, the preauthentication fails, and the KDC has a key of the same keytype

in the database, then the e-data for the error KDC_ERR_PREAUTH_FAILED will be present containing the kdcSalt value encoded as an OCTET STRING.

As a performance optimization, the kdcSalt can be locally stored on the workstation along with the corresponding realm. If the local configuration is missing, or incorrect, the above error messages allow the client to find out the correct salt. Clients which are configured for symmetric key with a recoverable key keytype, attempt to preauthenticate with the salt from the local configuration as an input into their key, and if the local configuration is not present, the client does not use preauthentication.

The following new preauthentication types are proposed:

PA-PK-RECOVERY-USER-SUPPORTED	19
PA-PK-RECOVERY-DATA	20
PA-PK-RECOVERY-SET-SKEY-TO-PRINCKEY	21

The following new error code is proposed:

KDC_ERR_RECOVER_USER_NEEDED	67

We propose the following additional KDC database bits. The first database bit applies to all principals to indicate whether a principal should have a valid symmetric key in the database. The second bit applies to all principals that should have a valid symmetric key to indicate if the principal symmetric key is valid.

The second database bit is cleared when the KDC undergoes a recovery operation, and all principal secret keys are zeroized as well. The non-human principal keys are then regenerated when a request comes in, and the corresponding validity bits are set.

3.2 Protocol Specification

We now describe the recovery protocol. The recovery operation can be set into motion either because a compromise is detected, or as part of a periodic preventative operation. The KDC host operating system and KDC executable is restored from backup media, and the KDC is loaded

with a backup private/public key pair. The KDC database is also reloaded, and any secret keys are zeroized. The new KDC public key is signed by the appropriate authority and placed in the appropriate location and any necessary revocation steps are taken for the old certificate.

To complete the recovery process, the KDC will also notify users that need to update any shared secrets that are stored in the KDC database: a KRB_ERROR message with the error code KDC_ERR_RECOVERY_USER_NEEDED is sent in response to these user's AS_REQ messages that do not contain the PA-PK-RECOVERY-DATA preauthentication type, contain the PA-PK-RECOVERY-USER-SUPPORTED preauthentication type, when there is no valid symmetric key in the KDC database, but there needs to be one.

The following ASN.1 structure is encoded and placed into the error message e-data field (an OCTET STRING):

```
  UserRecoveryError ::= SEQUENCE {
        kdcSalt            [0] OCTET STRING,      -- to be hashed
                                                  -- with password
                                                  -- key K1
        kdcPublicValue     [1] SubjectPublicKeyInfo,
                                                  -- DH algorithm
        kdcPubValueId      [2] INTEGER,           -- DH algorithm
        nonce              [3] INTEGER OPTIONAL,  -- copy nonce
                                                  -- from AS_REQ
                                                  -- if paPkKeyRep
                                                  -- is not below
        paPkKeyRep         [4] OCTET STRING OPTIONAL
                                                  -- ASN.1 encoded
                                                  -- PA-PK-KEY-REP
                                                  -- from section
                                                  -- 3.4 of [2]
                                                  -- (encrypted
                                                  -- user private
                                                  -- keys)
        kdcCert            [5] SEQUENCE OF Certificate, OPTIONAL
                                                  -- cert chain
        changePassword     [6] BOOLEAN OPTIONAL,  -- user client
                                                  -- should use
                                                  -- change password
                                                  -- protocol if
                                                  -- present
  }
```

The purpose of the kdcPubValueId identifier in the error message is to enable the KDC to offload state to the client; the client will then send this identifier to the KDC in an AS_REQ message; the identifier allows the KDC to look up the Diffie Hellman private value corresponding to the identifier. Depending on how often the KDC updates its private

Diffie Hellman parameters, it will have to store anywhere between one and several dozen of these identifiers and their parameters.

The e-cksum field of the error message is not optional for this error code; it will contain the signature of the entire error message (as described in [1]: the signature is computed over the ASN.1 encoded error message without the e-cksum field, and then the signature is placed into the e-cksum field and the message is re-encoded.) The KDC will sign using the private half of its new active key pair.

Upon checking the KRB_ERROR message, the client obtains the user password and uses the kdcSalt to compute the new key K2 which is computed by SHA1 hashing the concatenation of the kdcSalt and the key K1 obtained from the user password. The result of the hash is converted into a DES key by truncating the last 12 bytes and fixing the parity on each of the first 8 bytes. The client then responds with a new AS_REQ message that includes both a PA-PK-RECOVERY-DATA padata-type preauthentication field along with a PA-PK-AS-REQ preauthentication field (see [2]). The PA-PK-RECOVERY-DATA must contain the newUserKey field. If the user's AS_REQ message passes the security checks, the KDC will reply with an AS_REP message that contains a PA-PK-AS-REP preauthentication field. The client will validate this message as described in [2]. (The procedure for 3DES needs to be defined).

We also define the PA-PK-RECOVERY-USER-SUPPORTED preauthentication field; it will accompany all AS_REQ messages from clients that support the recovery protocol that originate from user principals. The padata-value for this padata-type is an empty octet string.

If the KRB_ERROR message passes the security checks (the nonce should match the client AS_REQ nonce if the error message is a reply, and the KDC signature validates), the client replies to the KDC with an AS_REQ message containing the PA-PK-RECOVERY-DATA padata-type preauthentication field along with a PA-PK-AS-REQ preauthentication field (see [2]):

```
PA-PK-RECOVERY-DATA ::= SEQUENCE {
        kdcPubValueId      [0] INTEGER,           -- Copied from error
                                                  -- message
        newUserKey         [2] EncryptedData      -- only present in
                               OPTIONAL           -- reply to
                                                  -- KDC_ERR_RECOVERY_
                                                  -- USER_NEEDED error;
                                                  -- uses DH shared
                                                  -- key to encrypt the
                                                  -- new key K2.
        sigAll             [3] Signature          -- uses shared DH key
                                                  -- computed over
                                                  -- entire encoded
                                                  -- AS_REQ without
```

```
                                                    -- this field, then
                                                    -- re-encode message
                                                    -- with this field
}
```

The clientPublicValue field in the AuthPack structure must be filled in by the client (in the PA-PK-AS-REQ preauthentication field, since Diffie-Hellman is required).

Upon receiving this message from the client, the KDC then makes the normal PA-PK-AS-REQ validation and also checks that the sigAll seal is valid after computing the shared Diffie-Hellman key. We note that the KDC should use the ctime and cusec fields in the PA-PK-AS-REQ message to ensure that the client AS_REQ message is not a replay. (The KDC also checks that the kdcPublicKeyKvno is correct (that it is the current version), and uses the kdcPubValueId to look up its own Diffie-Hellman parameters).

The KDC now sends an AS_REP message with the PA-PK-AS-REP preauthentication fields. The client should validate this message (including the normal PA-PK-AS-REP checks).

If the changePassword flag was present in the KDC error message, the client should immediately obtain a change password service ticket and use the protocol in [3] to change the user password. This option is useful to support an environment where both non-recovery capable and recovery capable clients exist. Since multiple keytypes will exist on the KDC for a given user, the change password protocol password field is the raw user inputted password; the KDC is responsible for deriving the appropriate keys from this password. In particular, any change password requests should result in the recoverable keytypes being derived by the RFC 1510 string to key transformation with salt and then hashing as described above using the kdcSalt value.

4. Acknowledgement

This work was previously published as part of draft-ietf-cat-kerberos-pkinit-02.txt while the author was employed at Cybersafe Corporation, 1605 NW Sammamish Rd., Suite 310, Issaquah, WA 98027. Thanks to John Wray, Mark Davis, and the CAT working group for some valuable suggestions on how to improve the draft.

5. Bibliography

[1] J. Kohl, C. Neuman. The Kerberos Network Authentication Service (V5). Request for Comments 1510.

[2] B. Tung, C. Neuman, J. Wray, A. Medvinsky, S. Medvinsky, M. Hur, J. Trostle. Public Key Cryptography for Initial Authentication in Kerberos. ftp://ds.internic.net/internet-drafts/

draft-ietf-cat-kerberos-pkinit-07.txt

[3] M. Horowitz. Kerberos Change Password Protocol.
ftp://ds.internic.net/internet-drafts/
draft-ietf-cat-kerb-chg-password-02.txt

6. Expiration Date

This draft expires on May 23, 1999.

7. Authors' Addresses

Jonathan Trostle
Cisco Systems
170 W. Tasman Dr.
San Jose, CA 95134

Email: jtrostle@cisco.com, jtrostle@world.std.com