Breaking (but not yet Fixing) PKINIT-25

May 1, 2005

1 Overview

The attack is described in detail in Sec. 2. In brief, the intruder inserts herself into the usual Kerberos 5 message flow using non-Diffie-Hellman public key authentication. This allows the intruder to obtain credentials under her own name, but she uses the nonces, *etc.*, from the client's requests in doing so; this allows the intruder to forge responses to the clients requests that include the appropriate signatures. At the end of the sequence, the various servers know that they have interacted with the intruder (who must be a legal user for this attack to work), the client believes she has authenticated herself to the servers, and the intruder knows all of the freshly generated keys shared by the client and the various servers. As a result, the intruder may impersonate the application server in interacting with the client; alternatively, if the client attempts to interact with the server in a way that the intruder is allowed to, the interaction will proceed and may be monitored by the intruder (who knows the associated session key).

This attack is possible because the servers never sign the name of the client they believe they are interacting with in a way that is visible to the client herself. (The servers do encrypt the name of the client for whom they are granting credentials—in this case, the intruder's name—but this is inside a ticket which is opaque to the honest client.)

Also, there appears to be a variant of this attack in which the intruder tampers with the AS exchange messages and then simply impersonates the servers in all subsequent exchanges without interacting with them. (We have not examined this as closely, but it works at an abstract level of formalization.) While the intruder may impersonate the application server as before, in this case the client does not share a key directly with this server.

2 The Attack

Figure 1 shows the message flow for the possible attack described here. In the remainder of this section, we describe the message fields shown and omitted as well as the actions of the various participants in this message flow.

2.1 The AS exchange

The client C sends a `KRB_AS_REQ` message, which is intercepted by the intruder I. Tables 1 and 2 show our treatment of the various fields of this message; we believe that the omitted fields have no effect on whether this is an actual attack. In modeling C's request to K, we use $ASRB(C, T, n_1)$ to denote the `req-body` portion of the `KRB_AS_REQ` message. I intercepts this message and constructs a

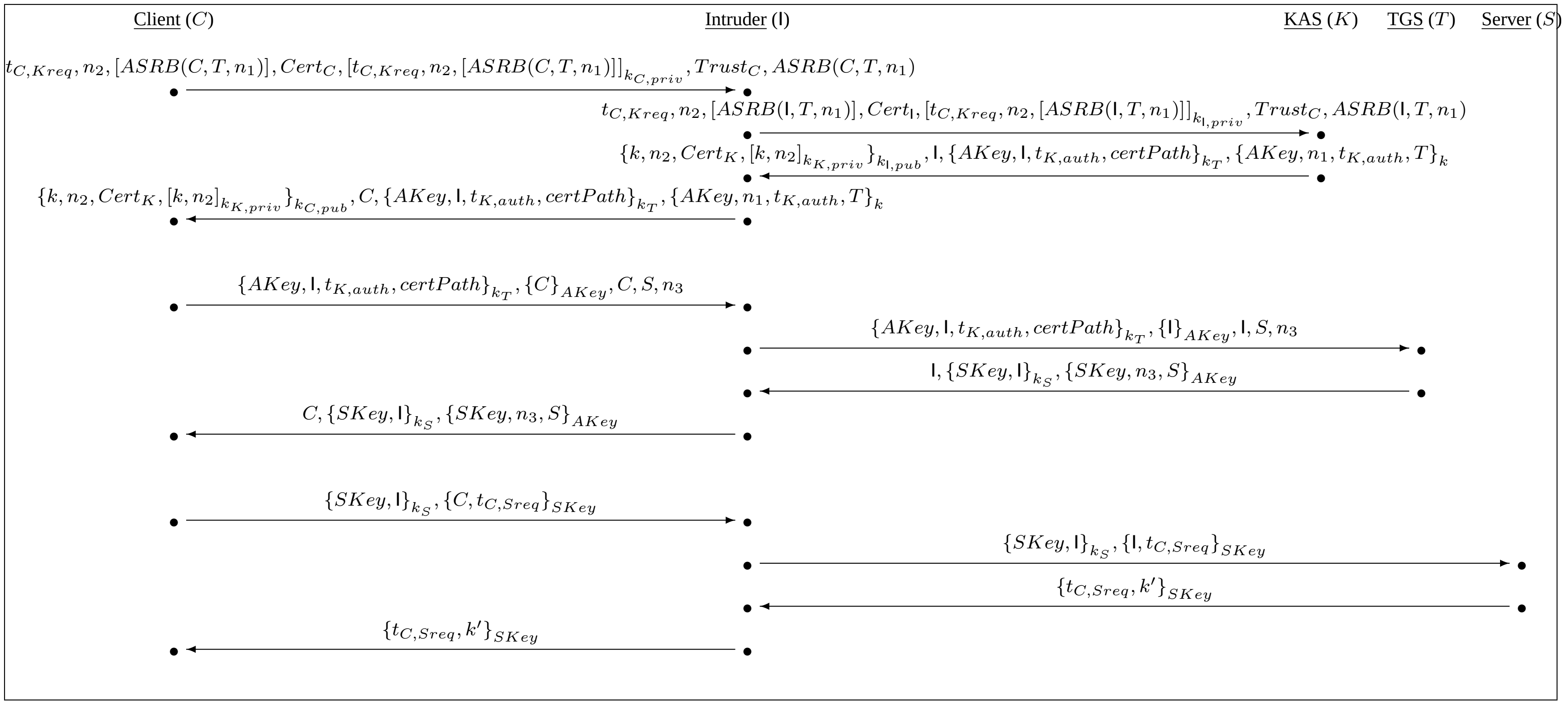
Client (C)
Intruder (I)
KAS (K)
TGS (T)
Server (S)
$t_{C,Kreq}, n_2, [ASRB(C,T,n_1)], Cert_C, [t_{C,Kreq}, n_2, [ASRB(C,T,n_1)]]_{k_{C,priv}}, Trust_C, ASRB(C,T,n_1)$
$t_{C,Kreq}, n_2, [ASRB(\mathsf{I},T,n_1)], Cert_{\mathsf{I}}, [t_{C,Kreq}, n_2, [ASRB(\mathsf{I},T,n_1)]]_{k_{\mathsf{I},priv}}, Trust_C, ASRB(\mathsf{I},T,n_1)$
$\{k, n_2, Cert_K, [k, n_2]_{k_{K,priv}}\}_{k_{\mathsf{I},pub}}, \mathsf{I}, \{AKey, \mathsf{I}, t_{K,auth}, certPath\}_{k_T}, \{AKey, n_1, t_{K,auth}, T\}_k$
$\{k, n_2, Cert_K, [k, n_2]_{k_{K,priv}}\}_{k_{C,pub}}, C, \{AKey, \mathsf{I}, t_{K,auth}, certPath\}_{k_T}, \{AKey, n_1, t_{K,auth}, T\}_k$
$\{AKey, \mathsf{I}, t_{K,auth}, certPath\}_{k_T}, \{C\}_{AKey}, C, S, n_3$
$\{AKey, \mathsf{I}, t_{K,auth}, certPath\}_{k_T}, \{\mathsf{I}\}_{AKey}, \mathsf{I}, S, n_3$
$\mathsf{I}, \{SKey, \mathsf{I}\}_{k_S}, \{SKey, n_3, S\}_{AKey}$
$C, \{SKey, \mathsf{I}\}_{k_S}, \{SKey, n_3, S\}_{AKey}$
$\{SKey, \mathsf{I}\}_{k_S}, \{C, t_{C,Sreq}\}_{SKey}$
$\{SKey, \mathsf{I}\}_{k_S}, \{\mathsf{I}, t_{C,Sreq}\}_{SKey}$
$\{t_{C,Sreq}, k'\}_{SKey}$
$\{t_{C,Sreq}, k'\}_{SKey}$

Figure 1: Message flow in the purported attack.

her own KRB_AS_REQ message using the parameters from C's message, replacing C's name with her own. The checksum $[ASRB(C, T, n_1)]$ in the pa-data is an unkeyed SHA1 checksum [], so I is able to generate the appropriate checksum $[ASRB(\mathsf{I}, T, n_1)]$ for the request containing her name. All data signed by C are sent unencrypted, so I may generate her own signatures in place of C's; she may also replace the certificates from C with her own set. The result is a well-formed KRB_AS_REQ message from I, although constructed using parameters taken from C's KRB_AS_REQ message.

When K replies with credentials for I, $AKey$ is encrypted under k, which is encrypted under $k_{\mathsf{I},pub}$, so the intruder learns both of the freshly generated keys. She then forges a KRB_AS_REP message to C using the signature $[k, n_2]_{k_{K,priv}}$ and the keys k and $AKey$; the part formerly encrypted under $k_{\mathsf{I},pub}$ is now encrypted under $k_{C,pub}$. The ticket-granting ticket is opaque to C, so I does not need to tamper with it (which would be impossible unless k_T is compromised). Tables 3 and 4 show our formalization of the KRB_AS_REP message.

2.2 The TGS exchange

The client C initiates the TGS exchange as usual. I intercepts her KRB_TGS_REQ message and replaces C's name with I's, both in the plaintext part of the message and in the authenticator; I can create the corresponding authenticator because she knows $AKey$. I sends this message to T, who replies to this well-formed request (from I) with credentials for I to use with S. In the KRB_TGS_REP message from T, I replaces her name with C's everywhere except in the ticket for S, which is opaque to C, and forwards the result to C, who now believes that she has successfully completed the TGS exchange with T and has a key $SKey$ for use with S. Note that I learns $SKey$ because it is encrypted under $AKey$.

2.3 The CS exchange

Finally, C sends a KRB_AP_REQ message to S. I intercepts this, replaces C's name with I's throughout, and forwards the result to S; as before, I knows the key (here $SKey$) used to construct the authenticator and so may make the necessary modifications. (She could also simply return $\{t_{C,Sreq}\}_{SKey}$ without involving S.) Finally, I intercepts S's reply to C. This message may contain a subsession key, labeled as k' in Fig. 1, which I learns because she knows $SKey$. Whether or not the subsession key is present, I forwards S's response to C without modification.

2.4 Effects

Once the CS exchange finishes as above, I knows $SKey$ and any subsession key k' that may also be shared between C and S. C believes that she has successfully completed an AS exchange with S and that they share $SKey$ (and, optionally, the subsession key k'). S believes that he has successfully completed the AS exchange with I—indeed, he has—and that they share the key $SKey$ (and, optionally, k').

I may impersonate the server S, responding to further messages from C directly. I may also monitor any traffic between C and S encrypted using $SKey$ (or k'). Note, however, that C cannot interact with S *under her own name* because S believes the compromised keys belong to I.

3 Preventing the Attack

Inspection of the fields that we have omitted from the various protocol messages suggests that no combination of these would prevent this apparent attack. We have given a fairly detailed model of the AS exchange when PKINIT is used. While the other exchanges are modeled with less detail, I knows all keys that C

believes she shares with the various servers. I can thus forge any data encrypted or signed by C in C's requests, and she can decrypt any data encrypted for C; there does not appear to be any information in the TGS or CS exchanges signed by the server and including C's name under the signature.

We believe this apparent attack would be prevented by K including the name of the client to whom he is replying in the data that he signs. The signature using $k_{K,priv}$ in the third ($K \rightarrow$ I) and fourth (I $\rightarrow C$) messages in Fig. 1 would then be $[\mathsf{I}, AKey, n_2]_{k_{K,priv}}$; C could detect the tampering with the messages if she were expecting $[C, AKey, n_2]_{k_{K,priv}}$ in the reply from K.

We have not, however, given a formal proof that this prevents this apparent attack; we are in the process of a more extensive formal verification of Kerberos 5 with PKINIT, and anticipate including in this analysis whatever changes are made to PKINIT in the event that intruder actions described here do indeed constitute an attack on the protocol.

Field Name	Basic
`pvno`	(omitted)
`msg-type`	(omitted)
`padata`	$t_{C,Kreq}, n_2, ck, Cert_C, [t_{C,Kreq}, n_2, ck]_{k_{C,priv}}, Trust_C$
`padata-type`	(omitted)
`padata-value`	(of type `PA-PK-AS-REQ`)
`signedAuthPack`	(of type `ContentInfo`)
`contentType`	(omitted)
`content`	(of type `SignedData`)
`version`	(omitted)
`digestAlgorithms`	(omitted)
`encapContentInfo`	$t_{C,Kreq}, n_2, ck$
`eContentType`	(omitted)
`eContent`	$t_{C,Kreq}, n_2, ck$
`pkAuthenticator`	$t_{C,Kreq}, n_2, ck$
`cusec`	$t_{C,Kreq}$
`ctime`	(merged with `cusec`)
`nonce`	n_2
`paChecksum`	$ck = [ASRB(C, T, n_1)]$
`clientPublicValue`	(omitted—DH only)
`supportedCMSTypes`	(omitted—optional)
`clientDHNonce`	(omitted—DH reuse only)
`certificates`	$Cert_C$
`crls`	(omitted—optional)
`signerInfos`	$[t_{C,Kreq}, n_2, ck]_{k_{C,priv}}$
`version`	(omitted)
`sid`	(omitted)
`digestAlgorithm`	(omitted)
`signedAttrs`	(partially merged into `signature`)
`signatureAlgorithm`	(omitted)
`signature`	$[t_{C,Kreq}, n_2, ck]_{k_{C,priv}}$
`unsignedAttrs`	(omitted)
`trustedCertifiers`	$Trust_C$
`kdcPkId`	(optional)

Table 1: Fields in the KRB_AS_REQ message.

Field Name	Basic
`req-body`	$ASRB(C, T, n_1)$
`kdc-options`	(omitted)
`cname`	C
`realm`	(omitted)
`sname`	T
`from`	(omitted)
`till`	(omitted)
`rtime`	(omitted)
`nonce`	n_1
`etype`	(omitted)
`addresses`	(omitted—optional)
`enc-authorization-data`	(omitted—optional)
`additional-tickets`	(omitted—optional)

Table 2: More fields in the KRB_AS_REQ message.

Field Name	Basic
`pvno`	(omitted)
`msg-type`	(omitted)
`padata`	$\{k, n_2, Cert_K, [k, n_2]_{k_{K,priv}}\}_{k_{C,pub}}$
`padata-type`	(omitted)
`padata-value`	of type `encKeyPack`
`encKeyPack`	(of type `ContentInfo`)
`contentType`	(omitted)
`contentField`	
`version`	(omitted)
`originatorInfo`	(omitted—optional)
`recipientInfo`	
`version`	(omitted)
`rid`	(omitted)
`keyEncryptionAlgorithm`	(omitted)
`encryptedKey`	(omitted)
`encryptedContentInfo`	
`contentType`	(omitted)
`contentEncryptionAlgorithm`	(omitted)
`encryptedContent`	$\{k, n_2, Cert_K, [k, n_2]_{k_{K,priv}}\}_{k_{C,pub}}$
`version`	(omitted)
`digestAlgorithms`	(omitted)
`encapContentInfo`	k, n_2
`eContentType`	(omitted)
`eContent`	k, n_2
`replyKey`	k
`nonce`	n_2
`certificates`	$Cert_K$
`crls`	(omitted)
`signerInfos`	
`version`	(omitted)
`sid`	(omitted)
`digestAlgorithm`	(omitted)
`signedAttrs`	(partially merged into `signature`
`signatureAlgorithm`	(omitted)
`signature`	$[k, n_2]_{k_{K,priv}}$
`unsignedAttrs`	(omitted—optional)
`unprotectedAttrs`	(optional)
`crealm`	(omitted)
`cname`	C

Table 3: Included fields for the KRB_AS_REP message.

Field Name	Basic
`ticket`	$\{AKey, C, t_{K,auth}, certPath\}_{k_T}$
`tkt-vno`	(omitted)
`realm`	(omitted)
`sname`	(omitted)
`enc-part`	$\{AKey, C, t_{K,auth}, certPath\}_{k_T}$
`flags`	(omitted)
`key`	$AKey$
`crealm`	(omitted)
`cname`	C
`transited`	(omitted)
`authtime`	$t_{K,auth}$
`starttime`	(omitted)
`endtime`	(omitted)
`renew-till`	(omitted)
`caddr`	(omitted)
`authorization-data`	$certPath$
`ad-type`	(omitted)
`ad-data`	$certPath$
`enc-part`	$\{AKey, n_1, t_{K,auth}, T\}_k$
`key`	$AKey$
`last-req`	(omitted)
`nonce`	n_1
`key-expiration`	(omitted)
`flags`	(omitted)
`authtime`	$t_{K,auth}$
`starttime`	(omitted)
`endtime`	(omitted)
`renew-till`	(omitted)
`srealm`	(omitted)
`sname`	T
`caddr`	(omitted)

Table 4: More fields for the KRB_AS_REP message.