

CMPUT 299 (B1) Winter 2008

Security in a Networked World

Authentication Protocols

yannis@cs.ualberta.ca

A Simple Challenge-Response Scheme

- ▶ Challenge-Response Authentication
 - ▶ B sends (random/unpredictable) challenge R and A has to reply $E_{K_{ab}}\{R\}$ to demonstrate it possesses K_{ab} , the shared secret.
- ▶ Weaknesses:
 - ▶ Authentication is not mutual and is susceptible to Man-in-the-Middle attacks.
 - ▶ If the subsequent communication is not protected (e.g., encrypted) the connection can be hijacked.
 - ▶ An eavesdropper can record R and $E_{K_{ab}}\{R\}$ for an off-line attack and/or construct a dictionary of $R \leftrightarrow E_{K_{ab}}\{R\}$ associations. (Size of R matters.)

Keys and Third Party Authentication

- ▶ Need for $n(n-1)/2$ keys for n participants
- ▶ A possible solution: introduce an intermediary (third party) that stores one secret key with each participant (say secret key K_A for participant A), for a total of n keys.
- ▶ Here's one scenario (for A to communicate with B):
 - ▶ A sends a request to the intermediary.
 - ▶ The intermediary verifies A 's identity and generates a unique session key that is sent to A & B securely.
 - ▶ A & B communicate using the session key.
- ▶ The intermediary must be always on-line.
- ▶ The intermediary can be a single point of failure.
- ▶ The intermediary is a highly sensitive system.

Kerberos Authentication

- ▶ Single log-on facility for collections of users and services under one administrative domain.
- ▶ The Kerberos model:
 - ▶ The client (C) has to convince the server (V) of its identity by demonstrating it possesses a ticket given to it by the intermediary (AS). The server's key is K_v
 - ▶ First attempt:

(1) $C \rightarrow AS:$ $ID_C \parallel P_C \parallel ID_V$

(2) $AS \rightarrow C:$ Ticket

(3) $C \rightarrow V:$ $ID_C \parallel Ticket$

A password in the “clear” (!) known to AS.

Network address of C to avoid impersonating C. AS knows the request came from AD_C and it indirectly informs V about this fact.

$Ticket = E_{K_v} \{ ID_C \parallel AD_C \parallel ID_V \}$

Kerberos Authentication (cont'd)

- ▶ Some notes:
 - ▶ The client (or anybody else for that matter) cannot tamper with the Ticket because it is encrypted with a key, K_v , known only to AS and the V.
 - ▶ The server is convinced because it can check the decrypted value of ID_C to be equal to the ID_C sent in the clear in step (3) and to determine whether ID_C is communicating from AD_C .
- ▶ Big holes:
 - ▶ The password is in the clear, so anyone can replay the request and receive a ticket and as long as they can spoof AD_C they can impersonate C.

Second Attempt

- ▶ Communicating the password is redundant! Let anyone request a “ticket” but make it impossible for them to make use of the ticket if they are not who they claim to be. That is, reduce the “password” to a claim of identity, i.e., just to ID_C
- ▶ Relieve the intermediary from having to issue tickets for each service. Just use it to provide a “meta-ticket” which will act as generic permission to generate service tickets as needed.
 - ▶ Decouple the ticket-granting server (TGS) from the service-granting tickets. (Use password only once.)

Second Attempt (cont'd)

► At log-on time:

(1) $C \rightarrow AS:$ $ID_C \parallel ID_{tgs}$

(2) $AS \rightarrow C:$ $E_{k_c}\{Ticket_{tgs}\}$

$$Ticket_{tgs} = E_{K_{tgs}}\{ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_1 \parallel Lifetime_1\}$$

K_{tgs} & K_c are the secret keys of TGS & C known to AS.

Restrict the lifetime of the ticket. The $Ticket_{tgs}$ can be reused in future contact with the TGS. Exposure to impersonating C is limited in time.

Second Attempt (cont'd)

► For each requested service:

(3) $C \rightarrow TGS: \quad ID_C \parallel ID_V \parallel Ticket_{tgs}$

(4) $TGS \rightarrow C: \quad Ticket_V$

$$Ticket_V = E_{K_V}\{ID_C \parallel AD_C \parallel ID_V \parallel TS_2 \parallel Lifetime_2\}$$

TGS decrypts $Ticket_{tgs}$ to determine its validity.

TGS will determine if access is allowed according to any applicable access policies.

K_V is known only to TGS and V.

Notice the “symmetry” of (1)+(2) & (3)+(4)

Second Attempt (cont'd)

- ▶ To access the service V :

$$(5) C \rightarrow V: \quad ID_C \parallel Ticket_V$$

- ▶ There is one serious problem remaining: an attacker using a spoofed AD_C address and replaying the tickets before their expiration.
- ▶ Another equally problematic prospect is the lack of mutual authentications. The servers can be fakes.

Kerberos V4

► At log-on time:

(1) $C \rightarrow AS:$ $ID_C \parallel ID_{tgs} \parallel TS_1$

(2) $AS \rightarrow C:$ $E_{K_c} \{ K_{c,tgs} \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2 \parallel Ticket_{tgs} \}$

$Ticket_{tgs} = E_{K_{tgs}} \{ K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2 \}$

Unique session key for subsequent exchanges between C and TGS. (Instead of AS sending it to TGS, TGS will receive it indirectly via C when a service ticket is requested.)

Kerberos V4 (cont'd)

► For each requested service:

(3) $C \rightarrow TGS: ID_V \parallel Ticket_{tgs} \parallel Authenticator_C$

(4) $TGS \rightarrow C: E_{K_{c,tgs}}\{K_{C,V} \parallel ID_V \parallel TS_4 \parallel Ticket_V\}$

$Authenticator_C = E_{K_{c,tgs}}\{ID_C \parallel AD_C \parallel TS_3\}$

$Ticket_V = E_{K_V}\{K_{C,V} \parallel ID_C \parallel AD_C \parallel ID_V \parallel TS_4 \parallel Lifetime_4\}$

In the eyes of the TGS it binds the sender of (3) to the identity of C, since only C could have used $K_{c,tgs}$ (and $K_{c,tgs}$ is known to TGS because it was just sent inside $Ticket_{tgs}$). The TS_3 is very short.

Unique session key for subsequent exchanges between C and V. (Instead of TGS sending it to V, V will receive it indirectly via C when the service is accessed.)

Kerberos V4 (cont'd)

► To access service V:

(5) $C \rightarrow V$: $\text{Ticket}_V \parallel \text{Authenticator2}_C$

(6) $V \rightarrow C$: $E_{K_{C,V}} \{TS_5 + 1\}$

$\text{Authenticator2}_C = E_{K_{C,V}} \{ID_C \parallel AD_C \parallel TS_5\}$

In the eyes of the V it binds the sender of (5) to the identity of C, since only C could have used $K_{C,V}$ (and $K_{C,V}$ is known to TGS because it was just sent inside Ticket_V). The TS_5 is very short.

Optional step for mutual authentication. Notice that V convinces C that it could read the TS_5 value, which means V possesses K_V .

Observations about Kerberos

- ▶ Authenticators are one-time “disposable” cryptographic evidence of one's identity. This is in contrast to the Ticket which is reusable.
- ▶ Note that strict timing synchronization is required. Typically, nodes must accept timestamps as current even though they may be off by a skew factor.
- ▶ The full Kerberos environment includes the notion of “realms” that usually describe administrative boundaries.
- ▶ Interoperating realms are implemented by sharing keys across servers of different realms.

Public Key Cryptography

- ▶ Basic problems solved by public key cryptography
 - ▶ No need to have a pre-established trusted communication path to exchange shared keys.
 - ▶ No need to have $O(n^2)$ shared keys for all possible pair-wise communications of n participants.
- ▶ The “decryption” key and the “encryption” key are not the same (but simply one reverses the effect of the other).
- ▶ Each participant generates a key pair (public,secret) and publishes/announces the public part.
- ▶ The intent is to use the key pair for a long period.

Uses of Public Key Cryptography

- ▶ Encryption (for confidentiality from A to B)
 - ▶ Assume K_b and K_b' are B's public & private key
 - ▶ A encrypts message with B's public key: $E_{K_b} \{M\}$
 - ▶ Only B can decrypt meaningfully: $M = D_{K_b'} \{E_{K_b} \{M\}\}$
- ▶ Signature (for authenticating message from A)
 - ▶ A encrypts M with its own private key: $E_{K_a'} \{M\}$
 - ▶ B decrypts it using A's public key: $M = D_{K_a} \{E_{K_a'} \{M\}\}$

For all public key cryptosystems we will discuss, it is the case that $D_{K_x} \{E_{K_x'} \{M\}\} = D_{K_x'} \{E_{K_x} \{M\}\} = M$

In fact even D and E will be the same function. (RSA)

Elementary Authentication Using Public Keys

- ▶ The contents of the messages could include all the trimmings we have seen in Kerberos (timestamps, ids, etc.). We describe just the skeleton protocol.
- ▶ Note that we will take the middleman (authentication server) out but we'll replace it with a nebulous notion of “advertising the public keys”

(1) $C \rightarrow V$: ID_C (claim to be “C”)

(2) $V \rightarrow C$: $E_{K_C}\{R\}$

(3) $C \rightarrow V$: $E_{K_V}\{R\}$

V selects random R and sends it encrypted with C's public key.

C decrypts V's message with its private key, extracting R, encrypts it then with V's private key. V decrypts with its private key and confirms it's R

Mutual Authentication

- ▶ Anybody could pretend to be V
 - ▶ On the previous slide, anybody (not just V) can generate (2), i.e., step (2) is not particular to V . Also a “fake” V can silently accept whatever is sent by C at step (3).
 - ▶ Add a “hoop” for V to demonstrate it is V (i.e., it knows the private key of V).

(1) $C \rightarrow V$: $ID_C \parallel E_{K_V}\{R_C\}$

C selects random R_C and sends it encrypted with V 's public key.

(2) $V \rightarrow C$: $E_{K_C}\{R_C \parallel R_V \parallel K\}$

V decrypts C 's R_C and responds encrypting it with C 's public key while also challenging it with a random R_V as well as a symmetric encryption (E) key K .

(3) $C \rightarrow V$: $E_K\{R_V\}$

K can be used in subsequent steps
(and it is a good idea for reasons of performance.)

Session Key

- ▶ Two mutually authenticated parties can exchange an encrypted session key to be used by a symmetric-key encryption algorithm.
 - ▶ Symmetric-key cryptography is computationally less demanding than asymmetric-key cryptography.
- ▶ If the purpose is agreeing on a session key, one can use the Diffie-Hellman key exchange scheme instead of RSA.
 - ▶ However, RSA can be used also for encryption (but not as efficient as symmetric-key) and for digital signatures.
 - ▶ Note that DH is exposed to MITM attacks.

Are MITM Attacks “Real” ?

Consider the tools that exist:

ettercap: MITM attacks on LAN subnets

monkey_jack: MITM attacks on 802.11 WLANs

dsniff: MITM attacks against SSL

... many others

Attack Against Public Keys

- ▶ How was the public key of a participant acquired?
 - ▶ Some “directory service” provides it (but do we trust such directory service?) or piggybacked on the same message used to set up an authenticated session.
 - ▶ If the participants are not trusted to begin with (that's why the authentication is necessary!) why trust them to advertise the legitimate/correct public key?
 - ▶ What “legitimizes” a public key: a signature by a third party that we already trust.
 - ▶ However a signature can be checked if we know with certainty the public key of the third party that signed the public key.
 - ▶ A chicken and egg problem.

Certification Authorities

- ▶ CAs are the entities that sign public keys.
 - ▶ A “certificate” is the signed public key + some additional relevant info (identity info, expiration, etc.)
- ▶ CAs seem to be equivalent to key distribution center (KDC = AS + TGS) of Kerberos. But are they the same thing?
 - ▶ What would happen if the system was compromised (KDC's keys exposed vs CA's public key exposed)?
 - ▶ Availability of the KDC vs. availability of the CA, i.e. what happens when the corresponding servers are unavailable?
 - ▶ Can both types of entities be isolated behind e.g., a firewall, or some other similar perimeter defense?

Relation of Certification Authorities

- ▶ The public key of CA can be signed by another CA as a mean to vouch for its accuracy.
 - ▶ A Universal CA (yeah right)
 - ▶ Hierarchical relationship
 - ▶ The trust of a user to a public key gets translated to a trust to the top-most CA that has signed the chain of certificates.
 - ▶ Arbitrary relationship (a-la PGP)
 - ▶ Users assume the role of signing other users public keys. End users decide who to trust. (No need for CAs)
 - ▶ Multiple (parallel) CAs
 - ▶ Pre-configuration of which CAs are to be trusted. The pre-configuration usually is “bundled” with a software component (e.g., browser) and hence we end up trusting the integrity of such software!

(Firefox) Options → Advanced → Encryption → View → Certificates → Authorities

