Kerberos & IPSEC

- We’ve already seen SSL/TLS and how it’s used to secure two-party communications over the web.
- Today we look at two additional protocol suites in widespread use today to secure client-server and peer-to-peer transactions.
  - Kerberos
  - IPSEC

Kerberos

- Designed for single “administration domain” of machines & users: users, client machines, server machines, and the Key Distribution Center (KDC).
- No public key crypto.
- Provides authentication & encryption services.
- “Kerberized” servers provide authorization on top of the authenticated identities.

Kerberos History

- Designed as part of MIT’s Project Athena in the 1980’s.
- Migration to the IETF.
  - RFC 1510 (Kerberos v5, 1993).
- Used in a number of products.
  - Passport is essentially Kerberos done w/ client-side cookies over HTTP.

The Kerberos Model

- Clients
- Servers
- The Key Distribution Center (KDC).
- Centralized trust model.
  - KDC is trusted by all clients & servers.
  - KDC shares a secret, symmetric key with each client and server.
- A “realm” is single trust domain consisting of one or more clients, servers, KDCs.
Joining a Kerberos Realm

- One-time setup
  - Each client, server that wishes to participate in the realm exchanges a secret key with the KDC
  - If the KDC is compromised, the entire system is cracked
  - Because the KDC knows everyone’s individual secret key, the KDC can issue credentials to each realm identity

Kerberos Credentials

- Two types of credentials in Kerberos
  - Tickets
  - Authenticators
- Tickets are credentials issued to a client for communication with a specific server
- Authenticators are additional credentials that prove a client knows a key at a point in time
  - Basic idea: encrypt a nonce

Kerberos Credentials

- Two types of credentials in Kerberos
  - Tickets
  - Authenticators
- Tickets are credentials issued to a client for communication with a specific server
- Authenticators are additional credentials that prove a client knows a key at a point in time
  - Basic idea: encrypt a nonce

Picture of a Kerberos Realm

The Basic Kerberos Protocol

Assume client C wishes to authenticate to and communicate with server S

Phase 1: C gets a Ticket-Granting Ticket (TGT)
Phase 2: C gets a Ticket for S
Phase 3: C communicates with S

Protocol Definitions

Following Schneier (Section 24.5):

- \( C \) = client, \( S \) = server
- \( TGS \) = ticket-granting service
- \( K_x \) = x’s secret key
- \( K_{x,y} \) = session key for x and y
- \( \{m\}_Kx = m \) encrypted in x’s secret key
- \( T_{x,y} \) = x’s ticket to use y
- \( A_{x,y} \) = authenticator from x to y
- \( N_x \) = a nonce generated by x

The Basic Kerberos Protocol (1)

Phase 1: C gets a Ticket-Granting Ticket

1. C sends a request to the KDC for a “ticket-granting ticket” (TGT)
   - A TGT is a ticket used to talk to the special ticket-granting service
   - A TGT is relatively long-lived (~8-24 hours typically)
     - C \( \Rightarrow \) KDC: C, TGS, \( N_C \)

Sent in the clear!
### The Basic Kerberos Protocol (2)

**Phase 1: C gets a Ticket-Granting Ticket**

2. KDC responds with two items
   - The ticket-granting ticket
   - A ticket for C to talk to TGS
   - A copy of the session key to use to talk to TGS, encrypted in C’s shared key
     
     \[
     \text{KDC} \rightarrow \text{C}: \{T_s, T_{C,TGS}\}_K_{C,TGS}, \{K_{C,TGS}\}_K_{C,TGS}
     \]
   - Only the TGS can decrypt the ticket
   - C can unlock the second part to retrieve $K_{C,TGS}$

### The Basic Kerberos Protocol (3)

**Phase 2: C gets a Ticket for S**

3. C requests a ticket to communicate with S from the ticket-granting service (TGS)
   - C sends TGT to S along with an authenticator requesting a ticket from C to S
     
     \[
     \text{C} \rightarrow \text{TGS}: \{A_{C,S}\}_K_{C,TGS}, \{T_{C,TGS}\}_K_{C,TGS}
     \]
   - First part proves to TGS that C knows the session key
   - Second part is the TGT C got from the KDC

### The Basic Kerberos Protocol (4)

**Phase 2: C gets a Ticket for S**

4. TGS returns a ticket for C to talk to S
   (Just like step 2 above...)
     
     \[
     \text{TGS} \rightarrow \text{C}: \{T_{C,S}\}_K_{S}, \{K_{C,S}\}_K_{C,TGS}
     \]
   - Only S can decrypt the ticket
   - C can unlock the second part to retrieve $K_{C,S}$

### The Basic Kerberos Protocol (5)

**Phase 3: C communicates with S**

5. C sends the ticket to S along with an authenticator to establish a shared secret
   
   \[
   \text{C} \rightarrow \text{S}: \{A_{C,S}\}_K_{C,S}, \{T_{C,S}\}_K_{S}
   \]
   - S decrypts the ticket $T_{C,S}$ to get the shared secret $K_{C,S}$ needed to communicate securely with C

---

**Key Distribution Center (KDC)**

- C \rightarrow KDC: C, TGS, NC
- KDC \rightarrow C: \(T_{C,TGS}\)_K_{TGS}, \{K_{C,TGS}\}_K_{C,TGS}
- C \rightarrow TGS: \{AC,S\}_K_{C,TGS}, \{TC,TGS\}_K_{TGS}
- TGS \rightarrow C: \{TC,S\}_K_S, \{KC,S\}_K_{C,TGS}

**Ticket Granting Server (TGS)**

- C \rightarrow TGS: \{AC,TGS\}_K_{C,TGS}, \{TC,TGS\}_K_{TGS}
- TGS \rightarrow C: \{TC,S\}_K_S, \{KC,S\}_K_{C,TGS}
The Basic Kerberos Protocol (6)

Phase 3: C communicates with S

6. S decrypts the ticket to obtain the $K_{C,S}$ and
   replies to C with proof of possession of the
   shared secret (optional step)
   
   $$S \rightarrow C: \{\text{timestamp, opt. subkey}\}K_{C,S}$$
   
   Notice that S had to decrypt the
   authenticator, extract the timestamp & opt.
   subkey, and re-encrypt those two
   components with $K_{C,S}$

---

Picture of a Kerberos Realm

---

Thoughts on Kerberos...

- There’s no public key crypto anywhere in the
  base Kerberos spec, but you can modify the
  base protocols to use PK...
- Example: the initial “login” to the KDC could be
  done with public key for added security (e.g.
  PKINIT protocol)
- More on this from final project presentations...

---

PKINIT in Windows 2000

---

Thoughts on Kerberos...(2)

- Only the KDC needs to know the user’s
  password (used to generate the shared secret)
- You can have multiple KDCs for redundancy,
  but they all need to have a copy of the
  username/password database
- Only the TGS needs to know the secret keys
  for the servers
- You can split KDC from TGS, but it is common
  for those two services to reside on the same
  physical machine
Cross-realm trust is possible
- Just need to share a secret key between the KDCs for the two realms...
- Once accomplished, a user in realm A can get a ticket for a service in realm B

“Time” is very important in Kerberos
- All participants in the realm need accurate clocks
- Timestamps are used in authenticators to detect replay: if a host can be fooled about the current time, old authenticators could be replayed
- Tickets tend to have lifetimes on the order of hours, and replays are possible during the lifetime of the ticket

Password-guessing attacks are possible
- Capture enough encrypted tickets and you can brute-force decrypt them to discover shared keys
- (Another reason to use public key...)

It’s possible to screw up the implementation
- In fact, Kerberos v4 has had a colossal security breach due to bad implementations

Session keys were generated from a PRNG seeded with the XOR of the following:
- Time-of-day in seconds since 1/1/1970
- Process ID of the Kerberos server process
- Cumulative count of session keys generated
- Fractional part of time-of-day seconds
- Hostid of the machine running the server

The seed is a 32-bit value, so while the session key is used for DES (64 bits long, normally 56 bits of entropy), it has only 32 bits of entropy
- What’s worse, the five values have predictable portions
  - Time is completely predictable
  - ProcessID is mostly predictable
  - Even hostID has 12 predictable bits (of 32 total)
RNGs in Kerberos v4 (continued)
- Of the 32 seed bits, only 20 bits really change with any frequency, so Kerberos v4 keys (in the MIT implementation) only have 20 bits of randomness
- They could be brute-force discovered in seconds
- The hole was in the MIT Kerberos sources for seven years!

Ideal Protection: End-to-End
- Web security (SSL, https) does this over TCP
- IPSEC does this for any IP packet, at network layer
- Apps aware of/control SSL, don’t have to be for IPSEC

IPSEC
- IPSEC = IP (Internet Protocol) Security
- Suite of protocols that provide encryption, integrity and authentication services for IP packets
- Mandatory-to-implement for IPv6, optional (but available) for IPv4
- Consists of two main components:
  - IPSEC proper (encryption & auth of IP packets)
  - IPSEC key management

IPSEC Architecture
- Key management establishes a Security Association (SA) for a session
  - SA used to provide authentication/confidentiality services for that session
  - SA is referenced via a security parameter index (SPI) in each IP datagram header

IPSEC Operation
- Provides two modes of protection
  - Tunnel Mode
  - Transport Mode
- Protection protocols
  - Authentication and Integrity (AH)
  - Confidentiality (ESP)
  - Replay Protection
Tunnel Mode
- Encapsulates the entire IP packet within IPSEC protection
- Tunnels can be created between several different node types
  - Gateway to gateway
  - Host to gateway
  - Host to host

Three Types of Tunnels
- Tunnel Mode
- Transport Mode
- Host to Host
- Host to Gateway
- Gateway to Gateway

Tunnel Security vs End-to-End Security
- Tunnel Security
- End-to-End Security

Transport Mode
- Encapsulates only the transport layer information within IPSEC protection
- Can only be created between host nodes

Authentication and Integrity
- Verification of the origin of data
- Assurance that data sent is the data received
- Assurance that the network headers have not changed since the data was sent

Confidentiality
- Encrypts data to protect against eavesdropping
- Can hide data source when encryption is used over a tunnel
Replay Prevention
- Causes retransmitted packets to be dropped.

IPSEC Protection Protocols
- Authentication Header (AH)
  - Authenticates payload data
  - Authenticates network header
  - Gives anti-replay protection
- Encapsulated Security Payload (ESP)
  - Encrypts payload data
  - Authenticates payload data
  - Gives anti-replay protection

Authentication Header (AH)
- Authentication is applied to the entire packet, with the mutable fields in the IP header zeroed out
- If both ESP and AH are applied to a packet, AH follows ESP

IPSEC Authentication Header (AH) in Transport Mode
- Orig IP Hdr
- TCP Hdr
- Data
- AH Hdr
- TCP Hdr
- Data
- Integrity hash coverage (except for mutable fields in IP hdr)

IPSEC AH in Tunnel Mode
- Orig IP Hdr
- TCP Hdr
- Data
- IP Hdr
- AH Hdr
- Orig IP Hdr
- TCP Hdr
- Data
- New IP header with source & destination IP address
- Integrity hash coverage (except for mutable new IP hdr fields)

Encapsulated Security Payload (ESP)
- Must encrypt and/or authenticate in each packet
- Encryption occurs before authentication
- Authentication is applied to data in the IPSEC header as well as the data contained as payload
IPSEC ESP in Transport Mode

- Usually encrypted
- Integrity hash coverage

IPSEC ESP in Transport Mode

- ESP is IP protocol 50
- 22-36 bytes total

IPSEC ESP Tunnel Mode

- New IP header with source & destination IP address

IPSEC Key Management

- IPSEC Key Management is all about establishing and maintaining Security Associations (SAs) between pairs of communicating hosts

Security Associations (SA)

- New concept for IP communication
  - SA not a “connection”, but very similar
  - Establishes trust between computers
- If securing with IPSEC, need SA
  - ISAKMP protocol negotiates security parameters according to policy
  - Manages cryptographic keys and lifetime
  - Enforces trust by mutual authentication

Internet Key Exchange (IKE)

- Phase I
  - Establish a secure channel (ISAKMP SA)
  - Authenticate computer identity
- Phase II
  - Establishes a secure channel between computers intended for the transmission of data (IPSEC SA)
ISAKMP/OAKLEY

- Merge of two key management protocols
  - ISAKMP: Internet Security Association and Key Management Protocol
  - NSA-designed protocol to exchange security parameters (but not establish keys)
  - OAKLEY
  - Diffie-Hellman based key management protocol

ISAKMP/OAKLEY (2)

- What’s used today is a combination
  - ISAKMP provides the protocol framework
  - OAKLEY provides the security mechanisms

Main Mode

- Main mode negotiates an ISAKMP SA which will be used to create IPSEC SA
- Three steps
  - SA negotiation
  - Diffie-Hellman and nonce exchange
  - Authentication

Main Mode (Pre-shared Key)

- Main Mode (Kerberos)

- Main Mode (Certificate)
Quick Mode

- All traffic is encrypted using the ISAKMP Security Association
- Each quick mode negotiation results in two IPSec Security Associations (one inbound, one outbound)

Quick Mode Negotiation

- Header, IPSec Proposed SA
- Header, IPSec Proposed SA
- Header, Connected Notification
- Header, IPSec Selected SA
- Header, Hub

How It All Fits Together

Multiple IPSEC transforms may be wrapped successively around a single IP datagram
- Example: IPSEC transport sent over an IPSEC tunnel

Sending in Transport Mode

Sending in Tunnel Mode
### Receiving in Tunnel Mode

- Physical IP
- IPSec
- IP
- Application Data

### Receiving in Transport Mode

- Physical IP
- IPSec
- IP
- Application Data

---

### What is Network Address Translation (NAT)?

- Network Address Translation (NAT)
  - Dynamically modifies source address
  - Dynamically recomputes interior UDP/TCP checksums
- Port Address Translation (PAT)
  - Dynamically modifies TCP/UDP source address and port
  - Dynamically recomputes interior UDP/TCP checksums

---

### NATs Rewrite Address/Port Pairs

- **Kernel mode firewall hook**

---

### IPSEC AH and NAT

- Change in address or port will cause message integrity check to fail
- Packet will be rejected by destination IPSEC
- AH cannot be used with NAT or PAT devices

---

### IPSEC ESP and NAT

- Can change IP header in special cases only
  - Special TCP/UDP ignores pseudo header used in checksum calculation
  - Port information encrypted!
  - Can’t change ESP header because integrity hash coverage

---