

CS203 Gene's Class - Network Security Overview



LEC1a-background-203.pdf
PDF

this is the background lecture slide, read very well

Great — I've read the "**LEC1a-background-203**" lecture slides thoroughly. This slide deck offers a comprehensive, top-down introduction to **network security** based on Kurose & Ross's approach. Here's a breakdown of the key sections and what they emphasize:



Core Principles of Network Security

- **Confidentiality:** Keep messages secret using encryption.
- **Authentication:** Verify identity of sender/receiver.
- **Message Integrity:** Ensure the message is unaltered.
- **Access & Availability:** Prevent DoS, ensure legitimate access.



Threat Model & Attacks

- **Eve** (the adversary) can eavesdrop, spoof, hijack, inject, and jam.
- Attacks span all layers of the network stack — e.g., IP spoofing (network), SYN flooding (transport), phishing (human layer).



Cryptography Fundamentals

Symmetric Key Crypto:

- Same key used by sender and receiver (e.g., **DES**, **AES**, **3DES**).
- Modes like **ECB** (vulnerable to patterns) and **CBC** (better).

Public Key Crypto:

- Uses a public/private keypair.
- Example: **RSA** — encryption with public key, decryption with private key.
- Key exchange issues lead to **certification authorities** (CA).



Message Integrity & Authentication

- **Hash functions** (e.g., SHA-2, SHA-3).

- **MACs:** Combine secret keys with hashes for authentication.
- **Digital Signatures:** Encrypt hash with private key to prove authorship (non-repudiation).

Securing Applications

Secure Email (e.g., PGP):

- Combine **symmetric encryption** (for speed) with **public key encryption** (to share keys) and **digital signatures** (for integrity).

SSL/TLS:

- Three phases: handshake (auth + master secret), key derivation (4 keys), secure data transfer (encryption + MAC).

IPsec:

- Network-layer security: **AH** (auth only), **ESP** (auth + encryption).
- Uses **Security Associations (SA)**.

Wi-Fi Security:

- **WEP** is broken (due to IV reuse and weak crypto).
- **802.11i** (WPA2/WPA3) improves with dynamic key generation and EAP-based authentication.

Operational Defenses

Firewalls:

- **Stateless filters** (simple rules on IP, port).
- **Stateful filters** (track connection states).
- **Application gateways** (filter at application level).

Intrusion Detection Systems (IDS):

- **Deep packet inspection**, session correlation, pattern detection.

Would you like me to create a concise summary slide or mind map from this for presentation or revision purposes?





explain mac from slide 29

Absolutely. Let's break down **Message Authentication Codes (MAC)** as presented on **Slide 29** in a clear and intuitive way.

What is a MAC?

A **Message Authentication Code (MAC)** is a short piece of information (a "tag") that ensures:

-  **Message integrity:** the message hasn't been tampered with.
-  **Authenticity:** the message really came from someone who knows the shared secret.

Key Idea:

A MAC is like a **digital wax seal** — only people who know the secret can make it, and it proves the message is intact and legit.

How does it work?

Both **Alice and Bob** share a **secret key K** .

At the sender (Alice):

1. Take the message m .
2. Append the key K (or mix it in).
3. Compute a cryptographic **hash**: $H(m + K) \rightarrow$ this is the MAC tag.
4. Send both the message and the MAC:
 $m, H(m + K)$

At the receiver (Bob):

1. Bob receives m and $H(m + K)$.
2. Since Bob also knows K , he computes his own $H(m + K)$.
3. If Bob's result matches the received MAC tag \rightarrow the message is **valid**.

Real-World Example: HMAC

One of the most widely used MAC constructions is **HMAC**, which stands for **Hash-based Message Authentication Code**.

It uses:

- A cryptographic hash function like SHA-256 or SHA-1
- Inner and outer padding tricks for extra security

Formula:

plaintext

$$\text{HMAC}(K, m) = H((K \oplus \text{opad}) \parallel H((K \oplus \text{ipad}) \parallel m))$$

Where:

- K is the secret key
- m is the message
- opad and ipad are fixed padding values
- H is the hash function (e.g., SHA-256)

Why is this better than just using a hash?

Because a plain hash like $H(m)$ can't prove authenticity — **anyone can compute it**.

But $H(m + K)$ **requires knowledge of the key**, which only Alice and Bob share.

Would you like me to give a Python example to simulate MAC generation and verification?



explain slide 55 , 56

Sure — Slides **55** and **56** explain how to achieve **secrecy**, **authentication**, and **integrity** together in **secure email**, using a combination of **encryption**, **digital signatures**, and **hashing**.

Slide 55: Providing Authentication and Message Integrity

Goal:

Alice wants to send Bob a message m , and she wants Bob to:

- Verify **Alice** sent it (authentication)
- Be sure the message was **not modified** (integrity)

Solution: Digital Signature with a Hash

1. **Alice computes a hash** of the message:

$H(m) \rightarrow$ a short digest of the message.

2. **Alice signs the hash** using her **private key**:

$KA^-(H(m)) \rightarrow$ this is the digital signature.



3. **Alice sends** both:

- The original message m (in plaintext)
- The digital signature $KA^-(H(m))$

4. Bob verifies the signature:

- He computes $H(m)$ himself from the received message
- He decrypts the signature using Alice's **public key**:
 $KA^+(KA^-(H(m)))$
- If the two hashes match \rightarrow the message is authentic and unmodified.

This satisfies:

-  Integrity (hash matches)
-  Authentication (only Alice could have signed it)

Slide 56: Providing Secrecy, Authentication, and Integrity

Now Alice also wants the message contents to be **private** (confidential).

New Goal:

- Encrypt the message for secrecy
- Authenticate Alice as the sender
- Ensure message integrity

Solution: Combine 3 Keys!

1. **Hash the message** $\rightarrow H(m)$
2. **Sign the hash** with Alice's **private key** $\rightarrow KA^-(H(m))$
3. **Append signature** to message $\rightarrow [m, KA^-(H(m))]$
4. **Generate a new random symmetric key** KS
5. **Encrypt the whole bundle** with KS :
 $KS([m, KA^-(H(m))])$
6. **Encrypt the symmetric key** with Bob's **public key**:
 $KB^+(KS)$

Alice sends Bob:

- Encrypted message and signature: $KS([m, KA^-(H(m))])$
- Encrypted symmetric key: $KB^+(KS)$

Bob's Steps:

1. Decrypt $KB^+(KS)$ using his **private key** \rightarrow get KS
2. Use KS to decrypt the message and signature \rightarrow get $[m, KA^-(H(m))]$
3. Compute $H(m)$ from the plaintext

4. Verify signature using Alice's **public key**

Summary:

Goal	Technique
Secrecy	Encrypt with random KS , then encrypt KS with KB⁺
Integrity	Hash the message: H(m)
Authentication	Sign hash with KA⁻ (H(m))

Would you like a visual diagram or Python-like pseudocode to see this in action?



explain pgp authentication

Great question. Let's break down how **PGP (Pretty Good Privacy)** achieves **authentication** — specifically **sender authentication** — as described in **Slide 57**.

Goal of PGP Authentication:

Make sure that the **message really came from Alice**, and was **not tampered with**.

This is **authentication** and **integrity**, not secrecy.

Step-by-Step: How PGP Provides Authentication

PGP uses **digital signatures** to authenticate a sender:

1. Alice writes a message **m**

2. Alice hashes the message:

- Compute **H(m)** using a cryptographic hash function (e.g., SHA-1 or SHA-2)

3. Alice signs the hash:

- She **encrypts** **H(m)** using her **private key KA⁻**
- This produces the **digital signature**:
Signature = KA⁻ (H(m))

4. Alice sends both:

- The **plaintext message m**
- The **signature: KA⁻ (H(m))**

This bundle is called a **PGP-signed message**.

📥 How Bob Verifies Authentication:

1. Bob receives:

- Message m
- Signature $KA^-(H(m))$

2. Bob computes:

- His own hash of the message: $H(m)$

3. Bob decrypts the signature using Alice's public key KA^+ :

- $KA^+(KA^-(H(m))) \rightarrow$ should equal $H(m)$

4. If the hashes match:

- ✅ The message is **authentic** (came from Alice)
- ✅ The message has **not been tampered with**

🔄 Summary in Table Form

Step	Action	Purpose
1	Alice computes $H(m)$	Digest for integrity check
2	Alice signs: $KA^-(H(m))$	Proves Alice is sender
3	Alice sends $m, KA^-(H(m))$	Plaintext + signature
4	Bob computes $H(m)$	Verify original message
5	Bob decrypts signature w/ KA^+	Should recover same hash
6	If match, message is authentic and untampered	

📌 Example (PGP Signed Message)

```
plaintext

---BEGIN PGP SIGNED MESSAGE---
Hash: SHA1

Bob: My husband is out of town tonight. Passionately yours, Alice

---BEGIN PGP SIGNATURE---
Version: PGP 5.0
Charset: noconv
yhHJRHHgJGhg/12EpJ+lo8gE4vB3mqJ...
---END PGP SIGNATURE---
```

- The message is **plaintext**.
- The signature is at the bottom.

- Bob can use **Alice's public key** to verify this signature.

Would you like me to explain how **PGP handles secrecy** or how it **combines both authentication and encryption**?



explain slide 86

Sure! **Slide 86** discusses the **limitations of firewalls and gateways** — important for understanding what **they cannot protect against**, even though they're essential components in network defense.

Limitations of Firewalls and Application Gateways

Firewalls and gateways are powerful tools for enforcing security policies, but they're not perfect. Here's what the slide emphasizes:

1. IP Spoofing

- A firewall/router **cannot verify** if a packet *really* comes from the claimed source IP.
- Attackers can **forge (spoof)** source IP addresses and bypass simple rules.

 Example:

An attacker sends packets that appear to come from a trusted IP (e.g., a partner company), tricking the firewall into allowing them through.

2. Specialized Application Support Needed

- Each application that requires special treatment needs its own **application-level gateway**.
- Firewalls don't automatically understand app-level protocols like FTP, VoIP, or streaming unless specifically configured.

 Example:

You may need a custom proxy to allow secure video conferencing, while still blocking other traffic.

3. Client Software Must Be Aware

- Applications (e.g., browsers) must be **configured to talk through a proxy or gateway**.
- If the client doesn't know how to reach the gateway, security controls won't apply.

 Example:

Your web browser must be configured to use a proxy (e.g., set the HTTP proxy settings). If not, it bypasses the gateway entirely.

⚠️ 4. UDP Filtering is Crude

- Firewalls often use **all-or-nothing rules** for UDP because it's **stateless**.
- This makes it hard to selectively allow safe UDP traffic (e.g., DNS) while blocking malicious ones.

🧠 Example:

If you allow DNS, you might unintentionally allow port-hopping malware to tunnel through UDP.

⚠️ 5. Security vs. Usability Tradeoffs

- Strict firewall rules improve security but **reduce usability**.
- More open networks are easier to use but more vulnerable.

🧠 Example:

Allowing external SSH might help developers but expose the network to brute-force login attempts.

⚠️ 6. They're Not Bulletproof

- Even highly protected networks with firewalls and gateways can still be compromised.
- **Zero-day attacks, insider threats, or misconfigurations** can defeat firewall protections.

🧠 Example:

An employee clicks a malicious link inside the network — no firewall can stop that.

✅ Bottom Line:

Firewalls and gateways are essential, but:

They **can't guarantee complete security**, especially against **IP spoofing, user misconfiguration, or sophisticated application-layer attacks**.

Would you like a comparison between **firewalls, IDS, and IPS** to better understand where each fits?



explain intrusion detection system from the slide

Absolutely! Let's walk through what **Intrusion Detection Systems (IDS)** are, based on **Slide 87–88** of your lecture.

🧑‍💻 Intrusion Detection Systems (IDS) – Overview

🔍 What is an IDS?

An **Intrusion Detection System** is a security tool designed to **monitor and analyze network or system activity** to detect signs of:

- Attacks
- Malware
- Policy violations
- Suspicious behavior

Think of it like a **security camera** for your network — it doesn't block traffic like a firewall, but it watches everything closely and sounds an alarm when something fishy happens.

✗ Why Firewalls Alone Aren't Enough

Traditional **packet filters (firewalls)**:

- Only check **headers** (like IP addresses, ports, flags).
- Cannot **correlate multiple sessions** or detect **complex patterns**.

So they **miss attacks** that happen across multiple packets or involve tricky payloads.

✓ What IDS Can Do That Firewalls Can't

1. Deep Packet Inspection

- Looks inside **packet contents**, not just headers.
- Compares data to a **database of known attack signatures**, virus patterns, or malicious keywords.

 Example:

Detects if someone sends SQL injection payloads or shellcode, even if the IP is allowed.

2. Session & Traffic Correlation

- IDS can **track multiple packets/sessions** and **find relationships**.
- Useful for spotting coordinated or slow-moving attacks.

 Examples:

- **Port Scanning**: attacker probes many ports — each probe looks harmless, but together they reveal intent.
- **Network Mapping**: attacker tries to discover all internal devices.
- **Denial-of-Service (DoS)**: IDS can detect flooding attempts.

Where Is IDS Deployed?

As shown in **Slide 88**, IDS sensors are placed at key points:

- Between **Internet and firewall**
- Inside **DMZ (Demilitarized Zone)**
- Around **critical servers** (e.g., Web, DNS, FTP)
- Inside the **internal network**

This allows detection of both **external** and **internal** threats.

Types of IDS

Type	Description
Network-based IDS (NIDS)	Monitors network traffic in real-time (e.g., Snort)
Host-based IDS (HIDS)	Runs on individual machines, monitors system calls, logs, etc.
Signature-based	Matches known attack patterns (like antivirus)
Anomaly-based	Detects deviations from normal behavior (e.g., sudden traffic spikes)

What IDS Does When It Detects an Intrusion?

- **Alerts** the administrator (email, dashboard, log)
- May **log** the event for forensic analysis
- Some advanced IDS systems are part of **IPS** (Intrusion Prevention Systems) that **actively block** traffic

Summary

- IDS inspects **content + behavior**, unlike firewalls which just check headers.
- It detects **malicious patterns**, **correlates activity**, and **alerts admins**.
- Placing multiple IDS sensors across the network gives full visibility.

Would you like a table comparing **firewall vs IDS vs IPS** in terms of functionality and limitations?



explain slide 85

Certainly! **Slide 85** explains **Application Gateways**, a type of firewall that filters **network traffic at the application layer** (Layer 7 in the OSI model).

Let's break it down step-by-step:

What is an Application Gateway?

Also known as an **Application-Level Gateway** or **Proxy Firewall**, it is a security mechanism that:

- **Intermediates** between the internal network and the outside world.
- Filters traffic **based on application data**, not just IPs or ports.
- Enforces more **fine-grained control** compared to stateless/stateful firewalls.

Example Scenario from the Slide: Controlling Telnet Usage

Imagine an organization wants to **allow only specific internal users to access external Telnet servers**. Here's how the **application gateway** handles this:

How It Works — 3 Steps:

Step 1: User connects to the application gateway

- Users must **manually point their telnet client** to the gateway instead of the remote host.
- This ensures **all traffic passes through the gateway** first.

Step 2: Gateway authenticates the user

- If the user is **authorized**, the gateway establishes a new **telnet connection** to the destination (external server).
- The gateway sits **in between**, relaying data back and forth between the user and the external server.

Step 3: Router blocks all direct telnet traffic

- The perimeter router/firewall is configured to **block any telnet traffic that doesn't originate from the gateway**.
- This guarantees that **only traffic vetted by the gateway gets through**.

Key Characteristics:

Feature	Description
Protocol-aware	Understands application-layer protocols like HTTP, FTP, SMTP, Telnet
User-specific control	Can enforce access policies per user or group
Data inspection	Can examine content (e.g., filter commands, keywords, file uploads)
Acts as proxy	Terminates connection from the user and opens a new connection to the target

Benefits:

- Fine-grained control over **who can use what application**

- Can log **specific user actions**
 - More secure for **sensitive protocols** like Telnet or FTP
-

Limitations:

- Requires **client-side configuration**
 - Adds **latency** since it proxies all data
 - Needs **per-application rules** and **updates for new protocols**
-

Analogy:

Think of it like a **security guard** who checks not just your ID (IP address), but also **what you're saying and doing** — and only lets you through if you're both verified and behaving correctly.

Would you like a diagram showing how application gateways compare to traditional firewalls?