“Hello everyone, we are Team 11. We are presenting on Command Injection. My name is [Your Name], and our team also includes Jonghyeok Lee, Chanhang Lee, Nurnazihah Intan, and Pero Vidal Villalba. I’ll begin our presentation.

[Slide – What is Command Injection?]
First, what is a command injection attack?
It’s a vulnerability that occurs when unsafe user input is passed directly to the operating system. This allows attackers to run arbitrary commands, and in the worst case they can gain full control of the server.

[Slide – A Simple Example]
Let me show you a very simple example.
If there are no defenses in place and an attacker supplies the malicious input shown on the slide, the server could execute it and return sensitive files—such as the password file—in the response.

[Slide – Vulnerable Code Example]
This is easier to understand with code.
When code concatenates user input directly into a shell command like this, the application hands the entire string to the system shell. An attacker can then inject shell commands through that input.

So the core issue is: user input → insufficient validation → input executed by the OS shell. That’s the vulnerability model we must always keep in mind.

[Slide – Command Injection vs. SQL Injection]
Now that we understand the concept, let’s compare command injection with the commonly confused SQL injection.
Command injection targets the host operating system, executing shell commands such as ping or cat.
SQL injection targets the database, manipulating queries like SELECT or DROP. Both stem from poor input validation, but their targets and impacts are different.

[Slide – Types of Command Injection Attacks]
Finally, the main types of command injection:
In-band: the command output appears directly in the HTTP response.
Blind: the command executes but output isn’t shown; attackers confirm execution indirectly (for example, using time delays).
Out-of-band: the output is sent to an attacker-controlled channel, such as a DNS or HTTP callback.