What is Remote Code Execution and How Does It Work?

Key Takeaways

- RCE vulnerabilities turn simple software flaws into full system compromise
- Common vectors include deserialization flaws, memory corruption, SSRF chaining, and unsafe input handling.
- Public-facing applications, APIs, cloud workloads, and developer tooling are frequent RCE entry points
- Reducing RCE risk starts with secure engineering and cloud hygiene

Remote code execution (RCE) ranks as one of the most dangerous vulnerability classes in today's environment. It lets attackers execute arbitrary code on remote systems, typically without needing credentials or user involvement. From there, they roll out malicious code, extract sensitive data, ramp up privileges, and fan out across networks and cloud setups.

Recent threat intelligence shows that exploitation of public facing applications is a leading initial access vector. As per IBM X-Force 2025 Threat Intelligence Index report, roughly 30% of analyzed incidents involved exploitation of public facing applications, underscoring how exposed services and unpatched software drive real world breaches[1].

At the same time, cloud specific research notes that misconfigurations and vulnerabilities in cloud services and workloads remain among the most common cloud attack entry points, representing a significant share of initial attack vectors in recent breach studies.

This article explains what RCE is, how it works in traditional and cloud native stacks, what common exploit vectors look like, and why the risk is growing.

What Is Remote Code Execution?

Remote code execution is a vulnerability that lets an attacker make a target system **execute code** of their choosing, over a network connection. They don't need physical access. They often don't even need valid credentials.

Instead, they find a flaw in how the application or service handles **user input**, manages memory, or processes scripts.

Here's how a typical RCE scenario unfolds:

- The attacker sends crafted data—a web request, protocol message, or serialized object.
- The vulnerable code treats part of that data as executable instructions or uses it unsafely to execute arbitrary commands.
- The system runs malicious code with whatever privileges the vulnerable process has.

If that service runs with elevated rights, the attacker gains powerful control of the host right away. Even with restricted accounts, they can usually chain it to other flaws or misconfigurations to escalate privileges.

Common implementation patterns that create RCE risk include:

- Passing user input into shell commands or interpreters without strict sanitization, making it possible to inject malicious code.
- Deserializing untrusted data into complex object graphs that contain dangerous gadget chains.
- Using unsafe functions that allow buffer overflow attacks and other memory issues.
- Evaluating user supplied expressions in templating, search, or “rules” engines without sufficient constraints, effectively performing remote code evaluation.

RCE vs Arbitrary Code Execution (ACE)

“Arbitrary code execution” (ACE) describes any situation where the attacker can execute arbitrary code on a system. The term does not specify how the attacker achieved that capability.

RCE is a specific subset of ACE:

- ACE can be local or remote. A vulnerability triggered by opening a malicious document may be ACE, but not necessarily RCE.
- RCE always involves remote triggering across a network or trust boundary, without requiring a preexisting local foothold.

All RCE vulnerabilities are also ACE vulnerabilities. But not all ACE vulnerabilities can be exploited remotely. **This difference matters for prioritization:** RCE issues typically deserve faster patching because they allow direct external access to code execution and significantly increase the attacker’s ability to gain unauthorized access.

Why RCE Deserves Priority?

- RCE lets attackers run arbitrary code remotely on servers, endpoints, and cloud workloads.
- It is commonly used to deploy ransomware, crypto miners, backdoors, and to launch lateral movement, data exfiltration, and system compromise.
- Exploited public facing applications and cloud misconfigurations have been highlighted as top initial access vectors in recent threat and cloud security reports.

If you discover an RCE critical vulnerability on an exposed asset, treat it as an emergency. Patch or mitigate first, then validate detection coverage for both code execution exploits and post exploitation behavior.

Proactive Cloud Defense: Reduce RCE Exposure

- Automated inventory of IaaS/PaaS assets (AWS, Azure, GCP).
- Realtime detection of cloud misconfigurations and drift.
- CIS-based policies with built-in remediation guidance.

Download the Datasheet

How RCE Attacks Typically Unfold

Most RCE attacks follow a recognizable sequence. Understanding this path helps you design detection and response.

1. Reconnaissance and Target Selection

Attackers identify exploitable targets by:

- Scanning for specific products, versions, and exposed services associated with known RCE CVEs.
- Enumerating web applications, APIs, VPNs, and remote management interfaces.
- Probing cloud environments for exposed metadata endpoints, admin consoles, or misconfigured services.

Threat reports continue to emphasize exploitation of public facing applications as a major initial access vector, which lines up with this reconnaissance driven targeting.

2. Exploit and Payload Delivery

Once a target is chosen, the attacker delivers an exploit. Common delivery methods include:

- HTTP requests with malicious headers, query parameters, body content, or serialized objects.
- Protocol messages (SMB, RDP, custom protocols) that trigger parsing flaws.
- File uploads that embed executable scripts (web shells) in seemingly benign files.
- Malicious inputs into CI/CD variables or IaC templates that later get interpolated into commands or scripts.

The exploit is crafted so that, when processed, it diverts normal execution into a path that performs attacker controlled operations.

3. Remote Code Execution on the Target

If the exploit succeeds:

- The target process executes code controlled by the attacker, such as starting a shell, downloading and running a binary, or modifying critical files.
- The attacker typically tests basic commands to confirm code execution, context, and the attacker’s ability to move further (for example, checking OS version and network reachability).
- From there, they can stage additional tooling or implants.

At this moment, the technical vulnerability has become a live intrusion, and the focus must shift to containment and eradication.

4. Post Exploitation and Lateral Movement

After gaining code execution, attackers usually:

- Enumerate local credentials, configuration files, and secret stores.
- Discover other systems via internal DNS, network shares, or cloud APIs.
- Move laterally using collected credentials or tokens.
- Establish persistence through startup items, scheduled tasks, web shells, or implanted binaries.
- Exfiltrate data and, in many cases, deploy ransomware or other impactful payloads.

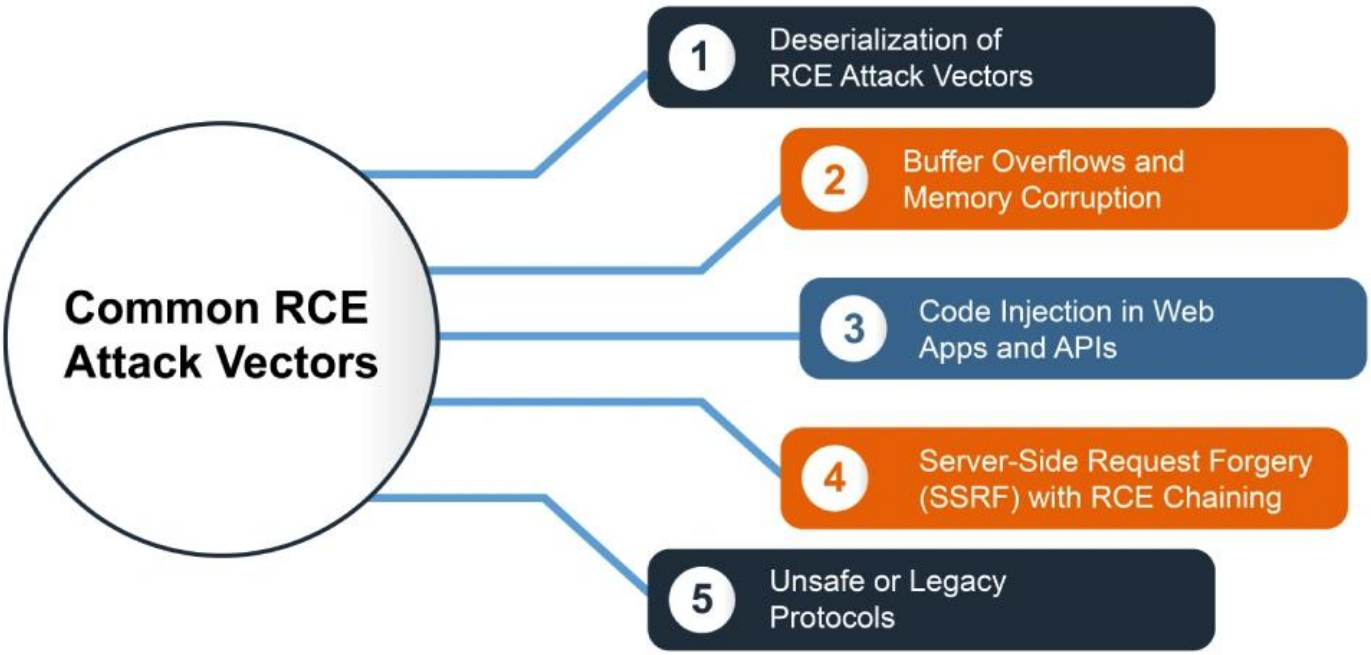
Controls that focus only on initial exploit signatures will miss much of this later activity. Full RCE defense has to address both initial exploitation and everything that comes after.

Common RCE Attack Vectors

Several technical patterns show up repeatedly in remote code execution attacks.

1. Deserialization of Untrusted Data

- Occurs when applications deserialize objects from untrusted sources.
- Attackers supply serialized payloads that cause the deserializer to instantiate gadgets with dangerous side effects and execute arbitrary commands.
- Frequently seen in older frameworks and protocols where deserialization was assumed to be safe.



2. Buffer Overflows and Memory Corruption

- Typical in native code that uses functions without proper bounds checking.
- Attackers overflow buffers, overwrite control structures, and redirect execution to injected shellcode or ROP chains.
- Common in protocol handlers, network daemons, and performance critical components.

3. Code Injection in Web Apps and APIs

- Happens when user input is directly concatenated into commands, queries, or expressions.
- Attacker controlled content reaches shells, SQL engines, templating logic, or scripting runtimes.
- Can turn a parameter tampering bug into full OS level RCE.

4. Server Side Request Forgery (SSRF) with RCE Chaining

- SSRF lets attackers make the server issue requests to internal endpoints they cannot reach directly.
- If those internal endpoints contain RCE capable functionality—such as admin APIs, metadata services, or script endpoints—SSRF becomes the first step in an RCE chain.

5. Unsafe or Legacy Protocols

- Older remote management protocols may include functionality that can be abused for RCE.
- Unpatched historical vulnerabilities in these protocols still exist in some environments.
- These services are often overlooked because they “have always been there,” but they increase the attacker’s ability to gain unauthorized access.

Famous RCE CVEs and What They Enabled

The table below summarizes representative RCE vulnerabilities and why they matter. It is illustrative and not exhaustive.

CVE ID Component / Ecosystem Exploit Vector (High Level) Typical Operational Impact CVE-2021-44228 Log4j (Log4Shell) JNDI lookup via crafted log messages Remote exploitation of many Java applications CVE-2017-5638 Apache Struts (Equifax breach) Crafted ContentType header in HTTP requests Webshell deployment and largescale data theft CVE-2025-11953 React Native Community CLI (mobile tooling) Malicious project configuration causing command execution in CLI RCE in developer environments and CI contexts CVE-2025-49844 Redis (cloudhosted) Abuse of serverside scripting and memory flaw RCE in Redis instances across cloud deployments runc breakout flaws (2025 series) Container runtime on Linux hosts Malicious container configuration and image content Escape from container to host in Kubernetes and similar platforms

These cases span traditional web stacks, dev tooling, cloud data stores, and container runtimes. The diversity highlights why RCE defense has to cover more than just web apps.

RCE in Cloud: Why the Risk Is Growing

Cloud and cloud-native adoption have accelerated sharply. CNCF Q3 2025 report on cloud-native development estimated that more than half of backend developers—around 56%—now build and run cloud-native workloads, with millions of such developers in the ecosystem[2]. As more organizations move workloads into containers, managed services, and multi-cloud architectures, the potential surface for cloud related RCE grows accordingly.

Several 2025 cloud and security studies emphasize that:

- Misconfigurations and insecure interfaces remain among the most frequently cited cloud threats, often appearing as a significant portion of initial attack vectors in breach data.
- Exploitation of public facing applications—including those running in cloud environments—was highlighted as a leading initial access vector in major 2025 threat intelligence reporting.

These data points reinforce a key theme: cloud RCE is not limited to a single product or service. It is a composite risk driven by exposed applications, misconfigurations, vulnerable dependencies, and complex control planes.

In Conclusion

Remote code execution is more than a software bug and can directly compromise the system. By allowing attackers to execute arbitrary code remotely, RCE vulnerabilities breakdown traditional security boundaries and give adversaries immediate leverage over servers, endpoints, and cloud workloads.

Understanding how RCE works and attackers use it to escalate privileges and lateral movement is important for effective defense. Knowing the common exploit vectors enable developers to establish secure engineering practices, disciplined patching, and strong cloud hygiene.

But this is just the beginning. Defending against RCE requires complete visibility across the entire lifecycle even across encrypted networks. To achieve this level of insight demands coordinated monitoring across networks, endpoints, and cloud environments.

Fidelis Elevate® combines network, endpoint, and deception capabilities for cross-domain visibility during intrusions like RCE. Fidelis CloudPassage Halo® separately addresses cloud and container risks.

Reference:

1. ^IBM X-Force 2025 Threat Intelligence Index | IBM
2. ^State Of Cloud Native Development Q3 2025