F

Fork Bomb: How Process Exhaustion Works, Recovery, and Prevention

A fork bomb recursively creates processes until a system becomes unavailable. Learn how to recognize it, recover safely, and enforce user, systemd, cgroup, and container limits.

PUBLISHER
Gridinsoft Support

UPDATED
Oct 8, 2026

READ TIME
6 min

CONTENTS

In this guide

01 How a fork bomb works
02 Signs of a fork bomb
03 Fork bomb vs similar failures
04 What to do during a fork bomb
05 What to check after recovery
06 Preventing fork bombs on Linux
07 Windows and macOS considerations
08 Frequently asked questions
09 References

A fork bomb is a denial-of-service program or command that creates new processes recursively until the system reaches a process, memory, or CPU limit. On Unix-like systems the name refers to the `fork()` process-creation model. Windows does not use the same fork call, but scripts or programs can cause the same resource-exhaustion effect by repeatedly launching new processes.

A fork bomb does not need to delete files to cause damage. Rapid process creation can make logins fail, prevent administrators from starting recovery tools, exhaust memory, and leave applications unable to save work. A restart may restore availability, but it does not explain who launched the code or prevent a repeat.

01 How a fork bomb works

1. One process starts one or more child processes.
2. Each child repeats the same process-creation action.
3. The number of processes grows exponentially or at another uncontrolled rate.
4. The operating system spends its resources scheduling, tracking, and allocating memory to them.
5. Normal programs cannot create processes or obtain enough CPU and memory to respond.

For safety, this article does not reproduce a runnable fork-bomb command. Copying a short example into a terminal can freeze the current session or an entire shared system.

02 Signs of a fork bomb

- The number of processes or tasks rises extremely quickly.
- Load average and CPU use spike while individual processes are short-lived.
- Shells report that they cannot fork or cannot allocate resources.
- New logins, remote sessions, or administrative commands fail to start.
- The desktop, terminal, or container becomes unresponsive without corresponding disk or network activity.

- Many processes share the same user, parent, command, script, or executable.

These symptoms are not unique to a fork bomb. A broken service restart loop, job scheduler, container orchestrator, build system, decompression job, or application bug can also create too many processes. Preserve the process tree and service logs when possible.

03 Fork bomb vs similar failures

Condition	Primary growth	Useful evidence
Fork bomb	Recursive child processes	Repeated parent-child pattern and rapidly rising PID count
Memory leak	Memory held by one or several long-running processes	Resident memory rises without matching process multiplication
Service crash loop	Repeated restarts by a supervisor	Service manager logs show exits and relaunches
Decompression bomb	Disk, memory, or CPU consumed while expanding data	Archive or parser process writes unexpectedly large output
Distributed denial of service	Inbound requests or traffic	Network telemetry shows many remote sources or high request volume

04 What to do during a fork bomb

1. **Do not start more ordinary applications.** Each attempted tool may need a process slot that is no longer available.
2. **Use an existing privileged session.** An already open root or administrator shell may still work when a new login cannot start.
3. **Identify the affected user or cgroup.** If tooling responds, inspect the process tree, owner, parent PID, unit, container, and command path.

4. **Stop the process group or workload.** Terminate the responsible service, container, user session, or cgroup as a unit. Killing children one at a time may lose the race because they keep reproducing.
5. **Isolate a shared host if necessary.** Pause new jobs and remote access so additional activity does not make recovery harder.
6. **Restart only when controlled termination fails.** A reboot restores process accounting but can destroy volatile evidence and interrupt unrelated workloads.

On a production server, follow the incident and availability procedure for that system. Forcing a power-off can corrupt active filesystems or databases. Prefer a controlled restart through an out-of-band console when the operating system no longer accepts commands.

05 What to check after recovery

- Which account, script, binary, service, container, or scheduled job started the process tree?
- Was it an intentional stress test, an accidental command, a software defect, or unauthorized activity?
- Did the same account make other changes before the outage?
- Were audit logs lost because storage or logging processes could not run?
- Did applications recover cleanly, or do databases and queues need consistency checks?
- Are process limits applied to the actual service and user that failed?

If the source is untrusted, treat the event as a security incident. A fork bomb can distract responders while another action occurs, and an attacker able to launch it may already have shell access.

06 Preventing fork bombs on Linux

Linux provides several layers of process control. Apply limits to the smallest practical scope and test them against normal peak demand.

- **User limits:** the `nproc` limit restricts how many processes a user can own. A temporary shell limit can be set with `ulimit -u`, while persistent policy is commonly managed through PAM limits.
- **systemd units:** `TasksMax=` limits tasks within a service or scope. A per-service limit is safer than relying only on a broad global value.
- **cgroup PID controller:** `pids.max` caps tasks in a cgroup and prevents one workload from consuming the host's entire PID space.
- **Containers:** set a PID limit as well as CPU and memory limits. Container isolation alone does not guarantee a process cap.
- **Service permissions:** run applications as separate, non-privileged accounts so one workload's limit does not affect unrelated services.

A limit that is too low causes legitimate failures; a limit that is almost as large as the host capacity provides little protection. Observe normal concurrency, allow justified headroom, alert on sustained approach to the limit, and test failure behavior.

07 Windows and macOS considerations

On Windows, use job objects, service isolation, application-control policies, and the resource limits provided by the relevant container or workload platform. Investigate repeated process creation through endpoint telemetry and parent-child process relationships.

macOS and other Unix-like systems support per-user resource limits, but configuration mechanisms differ. Managed environments should enforce limits through the supported service manager or device-management policy instead of depending on an interactive shell setting.

08 Frequently asked questions

Is a fork bomb a virus?

No. It does not have to infect files or spread between computers. It is a resource-exhaustion technique. Malware can include it, but an administrator or user can also trigger one accidentally.

Will a reboot remove a fork bomb?

It stops the current processes, but a scheduled task, startup service, container policy, or compromised account may launch them again. Find and correct the source before considering the incident closed.

Can a container fork bomb affect the host?

Yes, if the container has no effective PID limit. Namespaces separate process visibility, but the workload still consumes host resources. Use the cgroup PID controller and platform-specific container limits.

09 References

- SUSE: systemd resource controls and TasksMax
- systemd: Resource control
- Linux kernel: PID controller

NEED MORE HELP?

We are here when the guide is not enough.

Visit the Gridinsoft Support Center for current product guidance or submit a support request with the details of your case.

Open this guide online at support.gridinsoft.com