# **Dependency Confusion (DepConf) Attack (2021)**

### **1. Core Issue**

The Dependency Confusion attack was not a breach of a single vendor but a **systemic flaw in how package managers resolve dependencies**. Modern applications rely on both internal/private packages and public open-source packages. Attackers exploited the fact that package managers like **npm, PyPI, and RubyGems** sometimes prioritized **public packages over internal ones** if the same name existed. By publishing malicious “look-alike” packages to public registries with names matching internal corporate packages, attackers tricked builds into downloading and executing attacker-controlled code.

### **2. Who Was Attacked**

The technique was first demonstrated by security researcher **Alex Birsan** in 2021 as a proof-of-concept. He successfully executed the attack against **Apple, Microsoft, Tesla, Uber, and dozens of other major tech companies**. The organizations were not breached via direct intrusion but through automated build systems pulling in malicious dependencies.

### **3. Who Was Affected**

* **Large enterprises** that used internal/private package names overlapping with public registries.
* **Open-source ecosystems** (npm, PyPI, RubyGems) were indirectly affected, as the vulnerability exposed a design flaw in dependency resolution.
* Potentially thousands of developers and CI/CD pipelines could have been impacted if attackers had acted maliciously instead of researchers reporting it responsibly.

### **4. Exploit Chain Details**

1. **Reconnaissance** – Researcher guessed internal package names used by companies (via package.json, error logs, or code snippets).
2. **Public Registry Abuse** – Malicious packages with the same names were uploaded to public repositories (npm, PyPI, RubyGems).
3. **Automatic Resolution** – Build systems fetching dependencies resolved to the attacker’s package instead of the intended internal package.
4. **Execution of Malicious Code** – Once installed, attacker code ran with the same privileges as the build system, often exfiltrating sensitive data (environment variables, tokens).
5. **Proof-of-Concept Exfiltration** – Researcher exfiltrated hostnames, usernames, and internal build details to demonstrate the attack.

### **5. Prevention / Protection Steps**

* **Package Namespace Control**: Always publish internal package names to public registries to prevent typosquatting.
* **Scoped Packages**: Use organization-scoped names (e.g., @org/package) instead of plain names to avoid collisions.
* **Repository Priority**: Configure package managers to prioritize private repositories over public ones.
* **Audit Builds**: Regularly scan dependencies in CI/CD for unexpected external sources.
* **SBOMs (Software Bill of Materials)**: Maintain SBOMs to detect unauthorized dependencies.

### **6. Fixes & Vendor Response**

* npm, PyPI, and RubyGems released advisories and guidelines on mitigating dependency confusion.
* Many enterprises adopted scoped/private registries (e.g., Artifactory, Nexus).
* Bug bounty programs awarded **over $130,000** collectively to Alex Birsan for his responsible disclosure.

### **7. If No Fix Available**

* Immediately lock down dependency resolution settings in package managers.
* Mirror all dependencies internally and fetch only from trusted mirrors.
* Manually review build pipelines to detect unintentional pulls from public registries.

### **8. Reference Material**

* Alex Birsan – Dependency Confusion: How I Hacked Into Apple, Microsoft, and Dozens of Other Companies:  
   https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610
* GitHub Advisory – npm and PyPI Package Namespace Confusion:  
   https://github.blog/2021-02-12-avoiding-npm-namesquatting-dependency-confusion-supply-chain-attacks/
* Microsoft Security Response Center – Dependency Confusion Guidance:  
   https://msrc.microsoft.com/blog/2021/02/dependency-confusion-vulnerability-guidance/
* Sonatype Security Blog – Dependency Confusion in the Wild:  
   https://blog.sonatype.com/dependency-confusion-attacks-in-the-wild
* CISA – Open Source Software Supply Chain Compromise Alerts:  
   https://www.cisa.gov/news-events/alerts/aa21-073a
* PyPI Security Advisory – Namespace Confusion:  
   https://blog.pypi.org/posts/2021-02-19-pypi-now-supports-dependency-confusion-detection/

### **9. Further Reading**

* ENISA Threat Landscape for Supply Chain Attacks (2021):  
   https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks
* MITRE ATT&CK – Supply Chain Compromise (T1195):  
   https://attack.mitre.org/techniques/T1195/
* OWASP Software Supply Chain Security Guide:  
   https://owasp.org/www-project-software-supply-chain-security/
* OpenSSF Best Practices for Securing Package Registries:  
   https://openssf.org/working-groups/supply-chain-integrity/
* SANS Institute – Supply Chain Attacks via Dependency Confusion:  
   https://www.sans.org/blog/supply-chain-attacks-via-dependency-confusion/

### **10. Tooling**

* npm Scoped Packages – Use private scopes to prevent namespace collisions:  
   https://docs.npmjs.com/cli/v7/using-npm/scope
* PyPI Trusted Publishers & Package Verification:  
   https://docs.pypi.org/trusted-publishers/
* Sigstore / Cosign – Artifact signing and verification for packages:  
   https://sigstore.dev/
* in-toto – Supply chain metadata and provenance verification:  
   https://in-toto.io/
* npm Audit & GitHub Dependabot Alerts – Detect unexpected external dependencies:  
   https://docs.github.com/en/code-security/dependabot
* VirusTotal – Scan suspicious packages and URLs:  
   https://www.virustotal.com/