# **PyTorch Nightly Dependency Attack (2022)**

### **1. Core Issue**

The PyTorch incident was a **dependency confusion / supply chain compromise** that targeted users of the **nightly builds** of PyTorch, a widely used deep learning framework. Attackers uploaded a malicious package named torchtriton to the Python Package Index (PyPI), which conflicted with the internal dependency of the PyTorch nightly channel.

When users installed the nightly versions, their package manager (pip) pulled in the **malicious public version** of torchtriton from PyPI instead of the legitimate one. The malicious package exfiltrated sensitive information, including SSH keys, GPG keys, .gitconfig files, and system environment details.

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

The **PyTorch project** was the direct victim, as its nightly distribution system was abused. However, the attacker’s actual targets were:

* **Developers and researchers** installing PyTorch nightly builds.
* **Organizations** relying on nightly builds for testing or integration pipelines.

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

* Users of **PyTorch nightly builds** between **December 25–30, 2022** were at risk.
* Stable PyTorch users (standard releases from PyPI or Conda) were **not affected**.
* Affected victims could include AI researchers, academic institutions, and companies testing nightly builds in automated environments.

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

1. **Namespace Collision** – PyTorch nightly builds depended on a package named torchtriton, but this package was not yet published on PyPI.
2. **Malicious Upload** – Attackers registered torchtriton on PyPI and uploaded a malicious build.
3. **Automatic Resolution** – pip resolved the dependency from PyPI instead of the intended source, downloading the malicious version.
4. **Payload Execution** – The malicious package collected SSH keys, GPG keys, .gitconfig, system info, and transmitted them to attacker-controlled infrastructure.
5. **Detection & Response** – PyTorch maintainers discovered the issue on December 30, 2022, and issued advisories. Malicious packages were removed from PyPI.

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

* **Namespace Reservation**: Projects should proactively reserve critical dependency names on public registries (PyPI, npm, etc.).
* **Private Indexes**: Organizations should use private package mirrors and repositories for critical projects.
* **Dependency Pinning**: Pin dependencies to known hashes or signed versions instead of open version specifiers.
* **Registry Safeguards**: Package registries should implement namespace protection (preventing public takeover of names already used in large ecosystems).
* **Monitoring Dependencies**: Use dependency monitoring and SBOMs to identify malicious or unexpected new dependencies.

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

* PyTorch issued an **urgent security advisory** and instructed users to uninstall malicious versions and rotate potentially compromised keys.
* The project began publishing **signed wheels** to increase trust in artifacts.
* PyPI removed the malicious package and improved its internal monitoring for suspicious uploads.

### **7. If No Fix Available / Immediate Actions**

* **Uninstall compromised packages** immediately.
* **Rotate all SSH and GPG keys** used on affected systems.
* **Audit systems** for evidence of exfiltration or persistence.
* **Rebuild from clean sources** and ensure environments no longer trust unverified dependencies.

### **8. Reference Material**

* PyTorch Security Disclosure — Nightly Dependency Compromise:  
   https://pytorch.org/blog/compromised-nightly-dependency/
* BleepingComputer – PyTorch Dependency Attack Exposes Credentials:  
   https://www.bleepingcomputer.com/news/security/pytorch-nightly-builds-compromised-to-steal-credentials/
* MITRE ATT&CK – Dependency Confusion / Supply Chain Compromise (T1195):  
   https://attack.mitre.org/techniques/T1195/
* ENISA Threat Landscape – Software Supply Chain Attacks 2022:  
   https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks
* NIST – Software Supply Chain Security Guidance:  
   https://csrc.nist.gov/publications/detail/white-paper/2022/02/04/software-supply-chain-security-guidance/draft
* OpenSSF – Best Practices for Dependency Security:  
   https://openssf.org/

### **9. Further Reading**

* PyTorch Nightly Build Security Advisory and FAQ:  
   https://pytorch.org/security
* SANS Institute – “Securing Open-Source Dependencies in CI/CD”:  
   https://www.sans.org/blog/securing-open-source-dependencies/
* Google SLSA Framework – Build Integrity for Dependencies:  
   https://slsa.dev/
* ReversingLabs Report on Malicious PyPI Packages:  
   https://www.reversinglabs.com/blog
* GitHub Blog – Securing the Software Supply Chain:  
   https://github.blog/category/security/

### **10. Tooling**

* pip-audit – Detect known vulnerabilities in Python dependencies:  
  <https://pypi.org/project/pip-audit/>
* PyPI Trusted Publisher and Project Token Configurations:  
  <https://pypi.org/help/#trusted-publishers>
* Bandit – Static analysis for Python code security issues:  
   https://bandit.readthedocs.io/
* Sigstore / Cosign – Verifying build provenance for nightly packages:  
   https://sigstore.dev/
* Dependency-Track – Continuous monitoring of dependencies:  
   https://dependencytrack.org/
* Snyk – Supply chain scanning for Python projects:  
   https://snyk.io/