OCI Images as MCP Packaging: Supply Chain Security for AI Tools

Juan Antonio “Ozz” Osorio from STACKLOK

MCP Server Risks

Running an MCP server from npm may seem simple, but it exposes sensitive data. Understanding the **security implications** is crucial to safeguarding your systems against potential threats.

This Threat Continues to Evolve

POSTMARK-MCP

Malicious npm MCP server BCC'd all emails to attacker. ~1,600 downloads before removal. (Sep 2025)

MCP-REMOTE
CVE-2025-6514

Command injection, CVSS 9.6. 437K+ downloads affected. (Jul 2025)

SANDWORM_MODE

Worm-like malware via typosquatted AI packages. Harvested SSH keys. (Feb 2026)

Juan Antonio “Ozz” Osorio

Working @ STACKLOK

Created ToolHive: MCP server infrastructure with supply chain security

Background: security for OpenStack, Kubernetes, bare metal

"AI tooling should be secure by default, not secure by suffering"

What to Know

WHAT'S IN IT?

Understanding the Software Bill of Materials (SBOM) is crucial for transparency.

WHO BUILT IT?

Verify the publisher's identity through signature verification for trustworthiness.

HOW WAS IT BUILT?

Provenance attestation ensures the software was built from the expected source.

IS IT VULNERABLE?

Use vulnerability scanning results to identify potential risks in the software.

CAN I ENFORCE THIS?

Implement policy controls to ensure only verified packages are deployed.

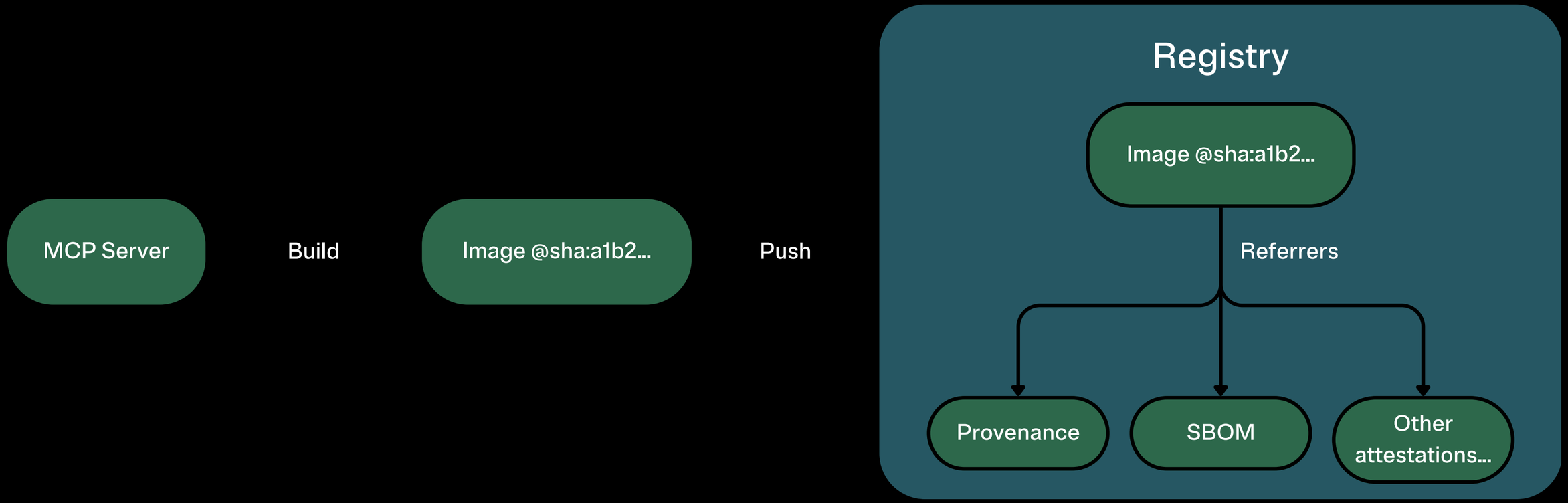
npm and PyPI Limitations

npm and PyPI provide some security features, but they lack comprehensive solutions like SBOMs and deploy-time enforcement, leaving significant gaps in supply chain security for developers.

OCI Images to the rescue!

OCI Images

- **Open Container Initiative**: the standard behind every container registry
- **Image**: your app + dependencies, identified by a cryptographic hash. It even supports multi-arch!
- **Registry**: where images live, and where security metadata attaches to them



What's in it?

The **npm audit** tool is limited in scope, only detecting vulnerabilities in application dependencies, while potentially ignoring critical issues hidden within OS packages, which can harbor numerous CVEs.

Who built it?

Understanding the **source of your npm packages** is crucial for security. A simple tarball can originate from uncertain sources, highlighting the need for reliable verification methods.

How was it built?

Understanding the **build process** is crucial for security. Knowing who signed isn't enough; we must verify the source repository to ensure the integrity of the software.

Can I enforce this?

Implementing policies can mitigate risks; consider using tools like Kyverno for enhanced security at deployment.

The Referrers API

The Referrers API provides a single API call to retrieve the **complete security metadata graph** for any image, integrating vital details like signatures, SBOMs, and scan results.

Capability	OCI	npm	PyPI
Signing	Sigstore's Cosign	Sigstore Provenance (GHA only)	Trusted Publishers
SBOMs attached to artifact	Yes (Referres API)	No	No
Attestations	Any in-toto predicate	Build provenance only	Publish attestation only
Vulnerability scanning scope	OS + App deps	App deps	App deps
Deploy-time enforcement	Kyverno / Sigstore policy controller	None	None
Metadata Graph	Referrers API	No equivalent	No equivalent

Demo Time!!

We’re building a repackaging pipeline! This demo showcases how to transform an MCP server package from npm into a verified OCI image through a streamlined re-packaging pipeline, ensuring security and integrity.

But wait... There's more...

Why only limit ourselves to MCP servers? What about skills?

OCI Artifacts

The **artifactType field** within OCI allows for the storage of various items beyond just containers, enabling a flexible approach to managing diverse software components and tools.

One Registry

The OCI model simplifies security by consolidating various artifact types under a single registry, ensuring a unified authentication method and streamlined security processes across all stored components.

Honest Limitations in Security

TOOL POISONING

Malicious package descriptions mislead users. OCI can sign annotations, but not solved today.

TOOL SHADOWING

Identical names mask malicious packages. Needs client-side verification, not packaging.

RUG PULLS

Developers suddenly change server behavior. Pinning by digest + signatures help mitigate.

Try These Actions

RUN GRYPE

Scan one MCP server to identify potential vulnerabilities in the software.

ADD COSIGN

Integrate cosign signing into your CI pipeline for enhanced security verification.

VALIDATE MCP SOURCES

Always check the origin of MCP servers before executing any commands with them.

ESTABLISH POLICIES

Implement security policies to enforce best practices for package management and deployment.

STAY INFORMED

Keep up with recent security incidents and updates related to MCP servers.

Resources

STACKLOK

ToolHive @ GitHub

ToolHive @ Discord

stacklok.com/ platform