SECURITY · TRUST · AGENTIC AI

Give AI the keys. *Keep the locks.*

An AI agent is not a chatbot. It writes and runs code, calls external tools, reaches out to the network, and holds your credentials to get work done. Every one of those powers is an attack surface. portablemind was built from the inside out to contain them — so your team can hand real capability to AI without handing over the building.

6	Per-tenant	Zero-trust
DEFENSE LAYERS AROUND EVERY AGENT ACTION	CRYPTOGRAPHIC ISOLATION BETWEEN CUSTOMERS	UNTRUSTED MODEL OUTPUT IS NEVER EXECUTED

THE REAL THREAT MODEL

Autonomy is the feature. It is also the risk.

The same abilities that make an agent useful are exactly what an attacker wants to hijack. Most "AI security" stops at the model. Ours starts where the agent actually touches your world.

01

It runs code

Agents generate and execute logic on the fly. Unsandboxed, that is a foothold on your servers.

02

It calls out

Fetching a URL an agent chose can be turned into a request to your internal network or cloud metadata.

03

It holds secrets

API keys, tokens, connection strings — the material an agent needs, and the material worth stealing.

04

It can be steered

Prompt injection hides instructions inside ordinary content, trying to turn a message into an exploit.

DEFENSE IN DEPTH

Six layers between the agent and everything it could touch.

No single control is trusted to be perfect. Each agent action passes through concentric layers of containment — so a failure in one is caught by the next, and no leaked token, injected prompt, or crafted URL reaches further than the layer it broke.

6 **Audit & identity**

Every access decision is authorized — and recorded in a tamper-evident trail.

5 **Encryption**

Secrets sealed at rest, everything encrypted in transit.

4 **Egress control**

Outbound requests can't reach your private network.

3 **Tenant isolation**

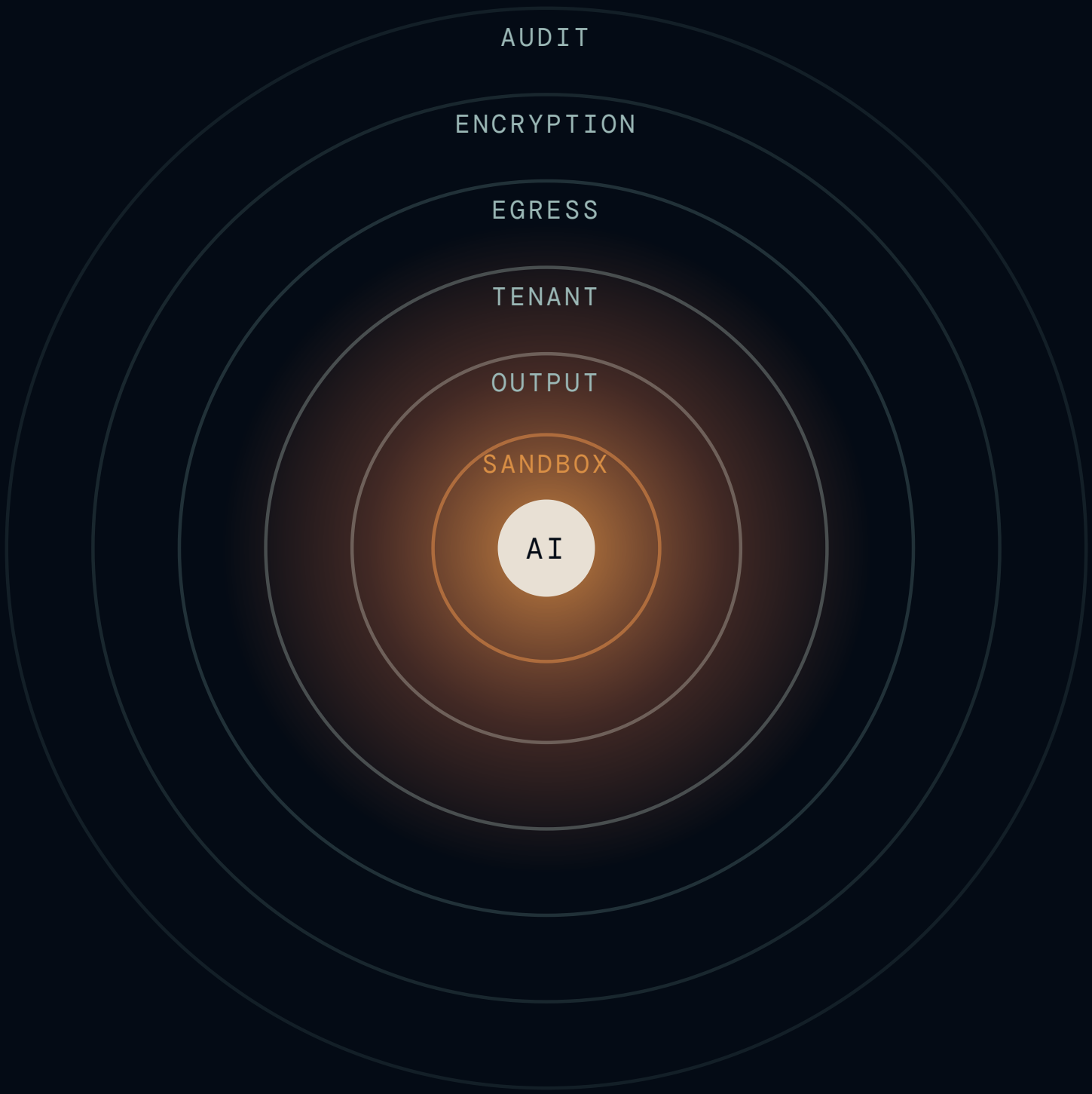
Your data and keys are cryptographically your own.

2 **Output containment**

Model output is rendered inert — never executed.

1 **Execution sandbox**

Agent code runs walled off from the host.



HOW EACH LAYER WORKS

Engineered controls, not security theater.

Layer 01

Execution sandbox

When an agent writes and runs code, it runs inside a hardened JavaScript isolate with **no filesystem access and no ambient access to environment variables or secrets** — only the specific, permission-checked operations we hand it. Outbound network calls are possible only through a **mediated, egress-controlled fetch**: every request is validated against an SSRF guard that blocks private, loopback, and cloud-metadata addresses, with an explicit per-tenant allow-list for approved hosts. Python skills are stricter still: agents can't author them at all. They run only vetted scripts from a registry, with **no shell, capped CPU and memory, and a hard kill on timeout**.

The blast radius of "the model wrote something malicious" is a sandbox that can't reach anything worth reaching.

V8 ISOLATE · NO AMBIENT FS / NET / ENV · REGISTRY-ONLY PYTHON · RESOURCE-CAPPED

Layer 02

Output containment

Model output is untrusted by default. Anything an agent generates — or that arrives inside a document, a ticket, or another user's message — is **rendered as inert content, never as executable markup**. Prompt injection can ask the model to emit an exploit; it can't make that exploit run in someone's browser.

Instructions hidden in content stay content. The boundary between data and code is enforced, not assumed.

NO RAW HTML EXECUTION · INJECTION-RESISTANT RENDERING · DATA ≠ CODE

Layer 03

Tenant isolation

Every customer is a cryptographically separate world. Sessions are signed with **per-tenant keys**, and every query is scoped to the tenant that owns it. A credential minted for one organization **cannot authenticate, read, or write anywhere else** — not through a clever ID, not through a forged header.

Multi-tenant convenience without multi-tenant leakage.

PER-TENANT SIGNING KEYS
SCOPED DATA ACCESS
NO CROSS-TENANT REACH

Layer 04

Egress control

When an agent fetches a URL, the request passes through a guard that **blocks private networks, loopback, and cloud-metadata endpoints** — closing the single most common path to stealing a cloud service account's credentials. The check re-validates on every redirect, so a public link can't quietly bounce to an internal address.

Need an agent to reach your own internal system? That's a deliberate, **per-tenant allowlist** you control — never an accident.

SSRF GUARD
METADATA ENDPOINT BLOCKED
PER-HOP RE-VALIDATION
TENANT ALLOWLISTS

Layer 05

Encryption

Integration credentials, OAuth tokens, and signing keys are **encrypted at rest with authenticated encryption**, partitioned away from ordinary data. Credentials the platform issues — API keys, reset and activation codes — are **never stored at all**: only their one-way digests are kept, and the raw value is shown exactly once. Everything on the wire is **TLS with strict transport security**. Sensitive values are filtered out of logs, so a debug trace never becomes a breach.

AES-GCM AT REST · HASHED CREDENTIALS, SHOWN ONCE · TLS + HSTS · SECRETS KEPT OUT OF LOGS

Layer 06

Identity & audit

Access is **capability-based and least-privilege**: an agent or user can do exactly what it's been granted and nothing more. Sessions are **short-lived and revocable**, accounts **lock after repeated failed logins**, and rate limits key on the identity being attacked — not just the attacker's IP. Inbound webhooks are cryptographically signed, replay-protected, and individually revocable.

Behind it all sits a **tamper-evident audit trail**: logins and failures, every token issued, every denial, every privilege change, and every touch of sensitive data land in an append-only, hash-chained ledger whose integrity is **verified nightly** and whose evidence copies export to write-once storage. And because **agents act through real, governed identities**, their trail is exactly as complete as any human's — filter by agent and see everything it authenticated, was refused, or accessed.

LEAST PRIVILEGE · BRUTE-FORCE LOCKOUT · SIGNED + REPLAY-PROOF WEBHOOKS · HASH-CHAINED AUDIT LEDGER · AGENTS AUDITED AS USERS

COMPLIANCE POSTURE

Built to the control families auditors ask about.

The architecture above maps directly onto the domains that HITRUST and SOC 2 assess — because we designed it that way, not to pass a checklist afterward.

Access control

Least-privilege capabilities, per-tenant isolation, and revocable sessions across every surface.

Encryption

Authenticated encryption at rest and TLS in transit, with disciplined key handling.

Transmission security

Signed, verified integrations and SSRF-guarded egress on every outbound call.

Audit & monitoring

A hash-chained, append-only audit ledger of logins, grants, denials, and sensitive-data access — humans and agents alike — verified nightly, with write-once evidence exports.

Authentication hardening

Brute-force lockout, identity-keyed rate limiting, hashed one-time credentials, and expiring, refreshable sessions.

Change & supply chain

Secret scanning and dependency-vulnerability gates run on every merge, in both application repos.

WHERE WE'RE HONEST

Security is a program, not a badge. We run **continuous, adversarial review** of our own platform, publish the controls above, and fix in the open — including the unglamorous work of tightening sessions, hardening egress, and closing gaps before an auditor finds them. If a control isn't real yet, we won't put it on a slide. That discipline is exactly why you can trust the ones that are.

Hand your AI real power. Sleep anyway.

See how portablemind contains agentic AI end to end — and what it would take to run your most sensitive work on it.

TALK TO US ABOUT SECURITY

↓ DOWNLOAD THIS BRIEF (PDF)