THREADSYNC — MAGIC RUNTIME

Magic Runtime — Security Model

February 2026 · v2.0 · magic.threadsync.io

Security Model

Magic Runtime is designed with security as a foundational principle. This document describes the threat model, isolation architecture, and security controls.

- Default-deny capabilities
- Process isolation
- Immutable audit logs
- Contract enforcement

Threat Model

Magic Runtime is designed to contain **first-party and AI-generated business logic** within a sandboxed environment. The security model assumes controllers may be buggy or attempt to exceed their declared permissions.

What the Sandbox Contains

THREAT	MITIGATION
Resource exhaustion (CPU/memory/disk)	cgroups enforcement with hard limits; timeout termination

THREAT	MITIGATION
Unauthorized database access	Capability broker validates table access against contract
Unauthorized network egress	Network namespace with explicit domain allowlist
Secret exfiltration	Secrets are scoped per-controller; no cross-controller access
Invalid input/output data	JSON Schema validation at request/response boundaries
Filesystem persistence	No persistent filesystem access; ephemeral temp only if declared

Explicit Limitations

The sandbox is **not designed** to run untrusted third-party code from unknown sources. It's designed to isolate your organization's business logic (including AI-generated code) within a controlled execution environment. For third-party integrations, use the `http:egress` capability to call external APIs.

Sandbox Architecture

Each controller execution runs in an isolated sandbox with multiple layers of containment.

Process Isolation

- **Separate process:** Each execution spawns a new process with restricted privileges
- **Seccomp filtering:** Syscall allowlist blocks dangerous operations (no raw sockets, no mount, no ptrace)
- **Namespace isolation:** PID, network, mount namespaces prevent cross-execution interference
- **User namespace:** Controllers run as unprivileged user inside container

Resource Limits (cgroups v2)

RESOURCE	DEFAULT	MAX CONFIGURABLE	ENFORCEMENT
CPU	0.5 cores	2.0 cores	cgroup cpu.max
Memory	256Mi	2Gi	cgroup memory.max (OOM kill)
Timeout	30s	300s	SIGKILL after deadline
Temp disk	0 (disabled)	100Mi	tmpfs with size limit

Network Isolation

Controllers have **no network access by default**. When `http:egress` is declared:

- Network namespace is created with egress proxy
- Proxy validates destination against declared `allow` domains
- Wildcard patterns supported: `*.stripe.com`
- All egress logged with destination, response code, latency

Egress Security Notes

Threat assumptions: The egress proxy validates at the DNS/SNI level. Protections include:

- **DNS rebinding:** The proxy resolves DNS at connect time and rejects private IP ranges (10.x, 172.16-31.x, 192.168.x, 169.254.x) unless explicitly allowlisted
- **SNI validation:** TLS connections verify the SNI hostname matches the declared allow domain
- **IP pinning:** After DNS resolution, the resolved IP is logged and used for the connection (no TOCTOU between resolve and connect)
- **Wildcard scope:** `*.example.com` matches subdomains only, not `example.com` itself. Declare both if needed

For environments requiring strict IP-based egress, configure `EGRESS_RESOLVE_MODE=static` and provide explicit IP mappings in the controller contract.

Capability Enforcement

Magic uses a **default-deny capability model**. Controllers must declare every external access in their contract. The runtime denies any undeclared access.

Capability Broker Architecture

1. Controller requests capability use (e.g., `self.db.query()`)
2. Request intercepted by capability broker
3. Broker validates against contract (table in allowlist?)
4. If denied: exception raised, request logged as violation
5. If allowed: request proxied to resource, logged for audit

Database Access Enforcement

How table allowlisting works: The capability broker intercepts all database access through a structured query layer. Controllers do not issue raw SQL.

- **Structured access only:** Controllers use `self.db.query(table, ...)` — the broker extracts the table identifier from the method call, not from SQL parsing
- **No raw SQL by default:** The DAL (Data Access Layer) translates structured queries to parameterized SQL. Controllers cannot construct arbitrary SQL strings unless the contract explicitly declares `db:rawsql` — which is disabled in production by default and not recommended
- **Allowlist check:** Before execution, the broker validates the target table against the contract's `tables` array. Unlisted tables are rejected with error `E2004`
- **Defense in depth:** For production deployments, we recommend also configuring per-controller database roles with schema-level `GRANT` permissions matching the contract allowlist

This approach is stronger than SQL parsing because table identity is carried as metadata, not extracted from potentially ambiguous SQL syntax.

Capability Matrix

CAPABILITY	GRANTS ACCESS TO	SCOPING
`db:read`	Structured read queries	Table allowlist
`db:write`	Structured write operations	Table allowlist
`db:rawsql`	Raw parameterized SQL (escape hatch)	Disabled in production by default
`http:egress`	Outbound HTTP/HTTPS	Domain allowlist
`secrets:read`	Secret values	Key allowlist
`queue:publish`	Message queue writes	Queue allowlist
`queue:subscribe`	Message queue reads	Queue allowlist
`cache:read`	Cache lookups	Key prefix
`cache:write`	Cache writes	Key prefix + TTL limit
`fs:temp`	Temporary files	Size limit

Audit Logging

All security-relevant events are logged to an append-only audit log with tamper-evident hash chaining.

Audit Immutability

Each audit entry includes a SHA-256 hash of the previous entry, forming a verifiable chain. For true immutability guarantees, configure an external WORM (Write-Once Read-Many) sink:

- **AWS S3 Object Lock** (Governance or Compliance mode)
- **Azure Immutable Blob Storage**
- **Append-only syslog** with remote attestation

Set `AUDIT_LOG_SINK` and `AUDIT_CHAIN_VERIFY=true` to enable hash chain verification on read.

Logged Events

- **Controller execution:** start, complete, error, timeout
- **Capability usage:** every db query, http request, secret access, queue operation
- **Capability violations:** denied access attempts with context
- **Resource limits:** exceeded thresholds, OOM kills, timeouts
- **Admin actions:** deploy, rollback, config changes (with actor identity)

Log Format

All logs are structured JSON with correlation IDs:

- `timestamp`: ISO8601 with microseconds
- `request_id`: Unique per-request identifier (propagated to egress)
- `controller`: Controller name and version
- `event`: Event type (execution_start, capability_use, etc.)
- `actor`: For admin actions, authenticated identity

SIEM Integration

Audit logs can be streamed to your SIEM (Splunk, Datadog, etc.) via syslog or webhook. Configure with `AUDIT_LOG_SINK` environment variable.

Authentication & Authorization

API Authentication

METHOD	HEADER	USE CASE
API Key	`X-API-Key: {key}`	Service-to-service, scripts
JWT Bearer	`Authorization: Bearer {token}`	User sessions, SSO integration
mTLS	Client certificate	High-security environments

Admin Plane Authorization

Administrative actions (deploy, rollback, config) require elevated permissions:

- **Role-based access control (RBAC):** Define roles with specific permissions
- **Separation of duties:** Deploy permission separate from secret management
- **Approval workflows:** Optional requirement for capability-expanding changes

SSO Integration

Enterprise deployments can integrate with identity providers:

- OIDC (Okta, Auth0, Azure AD, Google Workspace)
- SAML 2.0 (for legacy IdPs)
- Group-based role mapping

Production Hardening Checklist

Pre-Deployment Security Checklist

Enable process isolation mode

Set `CONTROLLER_EXECUTION_MODE=process` (not thread/inline)

Configure explicit controller allowlist

Set `CONTROLLER_ALLOWLIST=Controller1,Controller2` to prevent unauthorized deployments

Enable TLS termination

Configure Nginx with valid TLS certificate; force HTTPS redirects

Use strong secrets

Generate random `JWT_SECRET` (256-bit minimum); rotate API keys regularly

Restrict CORS origins

Set `ALLOWED_ORIGINS` to specific domains (not *)

Enable audit logging

Configure `AUDIT_LOG_ENABLED=true` and set retention policy

Set resource limits for production

Review controller contracts for appropriate CPU/memory/timeout values

Configure database connection security

Use TLS for database connections; prefer managed databases with encryption at rest

Secure Defaults

Magic ships with secure defaults that do not need to be changed:

- **No capabilities by default:** Controllers start with zero external access
- **Schema validation enabled:** Input/output validation cannot be disabled
- **Request IDs required:** All requests get unique identifiers for tracing
- **Audit logging on:** Security events always logged (destination configurable)

Vulnerability Disclosure

Reporting Security Issues

If you discover a security vulnerability in Magic Runtime:

1. **Email:** security@threadsync.io
2. Include: description, reproduction steps, impact assessment
3. We acknowledge within 48 hours
4. We provide status updates every 5 business days

Security Patch Policy

- **Critical:** Patch within 24 hours; immediate disclosure
- **High:** Patch within 7 days; coordinated disclosure
- **Medium/Low:** Patch in next scheduled release

> **Responsible Disclosure**
>
> We follow responsible disclosure practices. Reporters who follow our process will be acknowledged in release notes (unless they prefer anonymity).