IACADEMY

ENTERPRISE AGENT ENGINEERING

MÓDULO 05

Gobernanza de Agentes

Principal

iacedemy.com — 2026

MÓDULO 05

Gobernanza de Agentes

Nivel: Principal / Arquitecto

Autor: Alicia Sanz

Publicación: Julio 2026

Plataforma: iacedemy.com

Este material es parte del track Enterprise Agent Engineering de IAcademy.
Uso personal e intransferible.

> “Sin gobernanza, tus agentes son empleados sin contrato, sin auditoría y sin límites.”

4.1 Identidad y Acceso

Stack de Identidad

```
Keycloak (IdP)
    │
    ├── OIDC/OAuth2 → API Gateway
    │
    ├── JWT con claims:
    │   {
    │     "sub": "user-123",
    │     "tenant_id": "tenant-abc",
    │     "roles": ["agent-admin", "soc-analyst"],
    │     "agent_permissions": ["execute", "configure"],
    │     "budget_limit_eur": 100
    │   }
    │
    └── RBAC + ABAC → OPA Policy Engine
```

OPA Policy: Agent Permissions

```
# policy.rego
package agent.authz

default allow = false

# Solo admins pueden crear agentes
allow {
    input.action == "create_agent"
    input.user.roles[_] == "agent-admin"
}

# Cualquier rol puede ejecutar agentes de su dominio
allow {
    input.action == "execute_agent"
    input.agent.domain == input.user.domain
}

# Tools de riesgo HIGH requieren rol senior
allow {
    input.action == "execute_tool"
    input.tool.risk_level == "high"
    input.user.roles[_] == "senior-analyst"
}

# CRITICAL siempre requiere aprobación manual
deny {
    input.action == "execute_tool"
    input.tool.risk_level == "critical"
    not input.approval.approved
}

# Budget check
deny {
    input.action == "execute_agent"
    input.user.budget_used >= input.user.budget_limit
}
```

4.2 Gestión de Secretos

Vault Integration

```
import hvac
from functools import lru_cache

class SecretManager:
    def __init__(self, vault_url: str, role_id: str, secret_id: str):
        self.client = hvac.Client(url=vault_url)
        self.client.auth.approle.login(role_id=role_id, secret_id=secret_
id)

    def get_api_key(self, tenant_id: str, provider: str) -> str:
        path = f"tenants/{tenant_id}/api-keys/{provider}"
        secret = self.client.secrets.kv.v2.read_secret_version(path=path)
        return secret["data"]["data"]["key"]

    def rotate_key(self, tenant_id: str, provider: str, new_key: str):
        path = f"tenants/{tenant_id}/api-keys/{provider}"
        # Guardar versión anterior
        self.client.secrets.kv.v2.create_or_update_secret(
            path=path, secret={"key": new_key}
        )
        # La versión anterior se mantiene para rollback
```

Rotación Automática

```
async def rotate_all_keys(tenant_id: str):
    """Rotar todas las API keys de un tenant cada 90 días."""
    providers = ["openai", "anthropic", "qdrant"]
    for provider in providers:
        old_key = secret_mgr.get_api_key(tenant_id, provider)
        new_key = await provider_api.regenerate_key(old_key)
        secret_mgr.rotate_key(tenant_id, provider, new_key)
        log.info(f"Rotated {provider} key for {tenant_id}")
```

4.3 Auditoría

Schema de Audit Log

```sql
CREATE TABLE audit_log (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    timestamp TIMESTAMPTZ DEFAULT now(),
    tenant_id UUID NOT NULL,
    user_id UUID,
    agent_id UUID,
    action VARCHAR(50) NOT NULL,
    -- Qué modelo
    model_id VARCHAR(100),
    model_provider VARCHAR(50),
    -- Qué prompt
    prompt_hash VARCHAR(64),
    prompt_version VARCHAR(20),
    -- Qué tools
    tools_used JSONB,
    -- Coste
    tokens_input INTEGER,
    tokens_output INTEGER,
    cost_eur NUMERIC(10,6),
    -- Resultado
    success BOOLEAN,
    response_hash VARCHAR(64),
    -- Contexto
    ip_address INET,
    user_agent TEXT,
    session_id UUID,
    -- Compliance
    data_classification VARCHAR(20),
    gdpr_basis VARCHAR(50)
);

CREATE INDEX idx_audit_tenant ON audit_log(tenant_id, timestamp DESC);
CREATE INDEX idx_audit_agent ON audit_log(agent_id, timestamp DESC);
```

Middleware de Auditoría

```
class AuditMiddleware:
    async def __call__(self, agent_execution):
        start = time.time()
        try:
            result = await agent_execution()
            await self._log(agent_execution, result, success=True,
duration=time.time()-start)
            return result
        except Exception as e:
            await self._log(agent_execution, None, success=False, error=s
tr(e))
            raise

    async def _log(self, execution, result, success, duration=0, error=No
ne):
        await db.execute("""
            INSERT INTO audit_log (tenant_id, user_id, agent_id, action,
model_id,
                tokens_input, tokens_output, cost_eur, success,
tools_used)
            VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10)
        """, execution.tenant_id, execution.user_id, execution.agent_id,
            "execute", execution.model, execution.tokens_in,
execution.tokens_out,
            execution.cost, success, json.dumps(execution.tools_used))
```

4.4 Evidencias de Cumplimiento

Frameworks Soportados

```
COMPLIANCE_FRAMEWORKS = {
    "ens_alto": {
        "controls": 73,
        "evidence_types": ["policy", "config", "log", "report"],
        "audit_frequency": "annual",
    },
    "iso_27001": {
        "controls": 93,
        "evidence_types": ["policy", "procedure", "record", "report"],
        "audit_frequency": "annual",
    },
    "nis2": {
        "controls": 21,
        "evidence_types": ["policy", "incident_report",
"risk_assessment"],
        "audit_frequency": "biannual",
    },
    "ai_act": {
        "controls": 15,
        "evidence_types": ["risk_classification", "transparency", "human_
oversight", "data_governance"],
        "audit_frequency": "continuous",
    },
}
```

Generación Automática de Evidencias

```
async def generate_evidence(framework: str, control_id: str, tenant_id: s
tr) -> dict:
    """Generar evidencia automática para un control."""

    if framework == "ens_alto" and control_id == "op.acc.1":
        # Control: Identificación de usuarios
        users = await db.fetch("SELECT count(*), count(mfa_enabled) FROM
users WHERE tenant_id = $1", tenant_id)
        return {
            "control": "op.acc.1",
            "title": "Identificación de usuarios",
            "status": "compliant" if users["mfa_count"] == users["total"]
 else "non_compliant",
            "evidence": f"{users['mfa_count']}/{users['total']} usuarios
con MFA",
            "timestamp": datetime.now().isoformat(),
            "source": "automated_scan",
        }
```

4.5 Seguridad de Agentes

Threat Model

```
AMENAZA                    MITIGACIÓN
──────────────────────────────────────────────────
Prompt Injection           Input sanitization + guardrails
Indirect Injection         Content filtering en tool results
Prompt Leakage             System prompt obfuscation
Data Poisoning             Data validation + checksums
Tool Abuse                 Rate limiting + HITL + budget
Supply Chain (MCP)         MCP server verification + sandboxing
Model Extraction           Rate limiting + watermarking
Data Exfiltration          Output filtering + DLP
```

LLM Firewall

```python
class LLMFirewall:
    def __init__(self):
        self.input_rules = []
        self.output_rules = []

    async def check_input(self, prompt: str, context: dict) ->
tuple[bool, str]:
        for rule in self.input_rules:
            passed, reason = await rule.check(prompt, context)
            if not passed:
                return False, reason
        return True, "OK"

    async def check_output(self, response: str, context: dict) -> tuple[b
ool, str]:
        for rule in self.output_rules:
            passed, reason = await rule.check(response, context)
            if not passed:
                return False, reason
        return True, "OK"

# Reglas
class InjectionDetector:
    PATTERNS = [
        r"ignore\s+(previous|all|your)\s+instructions",
        r"system\s+prompt",
        r"you\s+are\s+now",
        r"pretend\s+to\s+be",
        r"DAN\s+mode",
    ]

    async def check(self, text, ctx):
        for pattern in self.PATTERNS:
            if re.search(pattern, text, re.IGNORECASE):
                return False, f"Injection pattern detected: {pattern}"
        return True, "OK"

class PIIDetector:
    async def check(self, text, ctx):
        # Detectar DNI, tarjetas, emails de otros tenants
        if re.search(r"\b\d{8}[A-Z]\b", text):
            return False, "DNI detected in output"
        if re.search(r"\b\d{4}[\s-]?\d{4}[\s-]?\d{4}[\s-]?\d{4}\b",
text):
            return False, "Credit card detected in output"
        return True, "OK"
```

4.6 Ejercicios

Ejercicio 1: OPA Policies

Escribe 5 políticas OPA para una plataforma de agentes: crear agente, ejecutar tool HIGH, acceder a datos de otro tenant, exceder budget, y desplegar en producción.

Ejercicio 2: Audit Trail

Diseña el schema de auditoría completo para tu plataforma. Incluye: quién, cuándo, qué modelo, qué prompt (hash), qué tools, coste, resultado, y clasificación de datos.

Ejercicio 3: LLM Firewall

Implementa un LLM Firewall con 5 reglas: injection detection, PII in output, cost limit, rate limit, y content classification.

Ejercicio 4: Compliance Dashboard

Diseña un dashboard que muestre estado de cumplimiento por framework (ENS, ISO, NIS2, AI Act) con: controles evaluados, gaps detectados, evidencias generadas, y próxima auditoría.
