Universidad Tecnológica de Bolívar

Inteligencia Artificial

Agente Inteligente de Ciberseguridad

Mauro Alonso González Figueroa, T00067622

Juan Jose Jiménez Guardo, T00068278

Revisado Por

Edwin Alexander Puertas Del Castillo

21 de septiembre de 2025

Índice

1. Introduccion

1.1. Caso de Estudio

La defensa cibernética es un conjunto de estrategias, tecnologías y procesos diseñados para proteger los sistemas informáticos, redes y datos contra amenazas cibernéticas, como ataques de malware, piratería informática, robo de datos y otros riesgos de seguridad. Estas medidas de defensa buscan prevenir, detectar, responder y recuperarse de ataques cibernéticos con el fin de garantizar la confidencialidad, integridad y disponibilidad de la información y los recursos digitales.

1.2. Resumen Ejecutivo

Este documento aborda el diseño e implementación de una ontología orientada a la defensa cibernética y su posterior aplicación en un agente inteligente. Primero se revisan los fundamentos teóricos de las ontologías: qué son, cómo se construyen actualmente y cuáles son las metodologías más utilizadas en su desarrollo. Luego, se analiza cómo estas ontologías pueden integrarse en sistemas inteligentes para mejorar la capacidad de detección, respuesta y resiliencia frente a amenazas cibernéticas.

El propósito central es proponer una ontología que sirva de base para dotar a un agente inteligente de un marco de conocimiento estructurado, que le permita tomar decisiones informadas en escenarios de ciberseguridad.

2. Modulo *PEAS*

El diseño del agente inteligente se describe formalmente mediante el modelo **PEAS** (Performance, Environment, Actuators, Sensors), el cual permite definir de manera estructurada los elementos fundamentales de su operación.

2.1. Performance (Medición del Desempeño)

Las métricas utilizadas para evaluar el desempeño del agente incluyen:

- Tiempo Medio de Respuesta (MTTR).
- Tiempo Medio Entre Fallos (MTBF).
- Efectividad de contención ante incidentes.
- Porcentaje de ataques prevenidos.
- Número de falsos positivos.
- Estimación del impacto económico evitado.

2.2. Environment (Entorno)

El agente opera en:

- Infraestructura de red corporativa.
- Servidores críticos y servicios en la nube.
- Dispositivos IoT y BYOD conectados a la red.
- Factores humanos y organizacionales, incluyendo políticas de seguridad, niveles de autorización y comportamiento del usuario.

2.3. Actuators (Actuadores)

Los actuadores del sistema incluyen:

- Bloqueo automático de tráfico malicioso.
- Aislamiento de dispositivos comprometidos.
- Módulo SOAR para respuesta orquestada (bloqueo, escalamiento, notificación).

- Generación de reportes ejecutivos y dashboards en tiempo real.
- Aplicación de parches y restauración de servicios.

2.4. Sensors (Sensores)

El agente integra sensores especializados para la detección de amenazas:

- Sensores de red para tráfico y anomalías.
- Monitores de integridad en sistemas y archivos.
- Antivirus y detección basada en firmas.
- Sensores basados en inteligencia de amenazas externa.
- Sensores de comportamiento de usuario (UEBA).

3. Ontología

3.1. Definicion

En IA y Web Semántica, una ontología es un modelo que define, con precisión y de forma formal, qué conceptos existen en un dominio, cómo se relacionan y qué reglas los gobiernan.

> "An ontology is an explicit specification of a conceptualization".
> *"Una ontología es una especificación explícita de una conceptualización"*
> Gruber, 2009

3.2. ¿De qué está hecha una ontología?

En la práctica, una ontología proporciona un vocabulario controlado junto con axiomas que otorgan significado computable a dicho vocabulario. Sus componentes típicos son:

- **Clases o conceptos:** Representan categorías o conjuntos de objetos dentro del dominio.

- **Propiedades o relaciones:** Describen cómo se vinculan los conceptos entre sí.
- **Individuos o instancias:** Son los elementos concretos que pertenecen a las clases.
- **Restricciones (axiomas):** Reglas que limitan o definen el comportamiento y las relaciones entre los elementos anteriores.

3.3. Lenguajes y estándares que la hacen operativa

Las ontologías se formalizan principalmente con OWL (Web Ontology Language) sobre el modelo de grafo de RDF. OWL aporta semántica formal para que los sistemas puedan inferir y verificar conocimiento, no solo almacenarlo; RDF define el modelo de tripletas (sujeto–predicado–objeto) que hace interoperable el intercambio de datos (W3C OWL Working Group, 2012).

Para validar calidad de datos frente a una ontología o esquema RDF, la recomendación *W3C SHACL* permite expresar "shapes" (condiciones) y chequear conformidad de grafos; existen borradores recientes que amplían su alcance (Knublauch & Kontokostas, 2017a).

3.4. Usos clave

1. Hablar el mismo lenguaje (interoperatividad semántica). Sirven para alinear significados entre equipos y sistemas distintos. Cuando varias aplicaciones comparten la misma ontología, se reduce la ambigüedad y se facilita el intercambio de información con sentido. Esta es, de hecho, una de las motivaciones fundacionales para "desarrollar una ontología".

2. Integrar datos heterogéneos. RDF/OWL permiten fusionar fuentes dispares (bases relacionales, logs, APIs, documentos) aun si los esquemas difieren, porque los recursos y relaciones se identifican con URIs y se normalizan como grafos de tripletas (RDF Working Group, 2014).

3. Razonar e inferir conocimiento implícito. Al tener semántica formal, se pueden derivar

hechos que no estaban explícitos (clasificación automática, detección de inconsistencias, implicaciones lógicas). OWL 2 fue diseñado justamente para habilitar interpretabilidad y razonamiento sobre contenido (Baader, 2003).

4. Búsqueda y recuperación semántica. Con una ontología, las consultas van más allá de palabras clave: explotan jerarquías y relaciones (por ejemplo, recuperar todos los incidentes que afectan activos críticos con vulnerabilidades). Este uso está expuesto en guías prácticas de desarrollo ontológico.

5. Validación y control de calidad de los datos. Mediante SHACL, las organizaciones definen reglas de calidad (cardinalidades, rangos, dependencias) y pueden auditar automáticamente si sus grafos cumplen políticas y estándares. Esto impacta directamente en la gobernanza de datos (Knublauch & Kontokostas, 2017b).

6. Reuso y extensibilidad del conocimiento. Las ontologías promueven reutilizar modelos consolidados (tiempo, unidades, ubicaciones) y extenderlos para nuevos proyectos, acelerando desarrollos y mejorando la consistencia entre equipos.

3.5. Ontología en la Inteligencia Artificial

En el contexto de la Inteligencia Artificial (IA), las ontologías se definen como modelos formales que representan de manera estructurada el conocimiento de un dominio específico. Estas permiten a los sistemas inteligentes interpretar, razonar y actuar sobre datos de forma más precisa y contextualizada.

Las ontologías facilitan la organización semántica de la información, permitiendo que los algoritmos de IA realicen inferencias lógicas, identifiquen patrones relevantes y generen respuestas adaptadas a las necesidades del usuario. Al establecer conceptos, relaciones y reglas dentro de un marco compartido, las ontologías mejoran la interoperabilidad entre sistemas, optimizan la recuperación de información y permiten simular procesos cognitivos humanos.

En aplicaciones prácticas, como buscadores semánticos, asistentes virtuales o sistemas expertos, las ontologías permiten que la IA comprenda el significado detrás de los datos, lo

que se traduce en una interacción más eficiente, personalizada y autónoma con el entorno digital (Martín et al., 2020).

4. Construcción de la Ontología

4.1. Metodología de Desarrollo

Para la construcción de la ontología de ciberseguridad, se adoptó un enfoque híbrido que combina la formalización lógica tradicional con herramientas modernas de razonamiento automatizado. En lugar de utilizar los lenguajes estándar como OWL (Web Ontology Language) o RDF (Resource Description Framework), se optó por implementar la ontología utilizando Z3 SMT Solver (Satisfiability Module Theories), lo que permite realizar razonamiento lógico de primer orden con verificación formal de consistencia.

Esta decisión metodológica se fundamenta en las siguientes ventajas:

- Expresividad lógica: *Z3* permite definir reglas complejas usando cuantificadores universales y existenciales
- Verificación automática: Capacidad de verificar satisfacibilidad y consistencia del modelo en tiempo real
- Integración nativa: Implementación directa en Python sin necesidad de parsers externos
- Rendimiento: Optimizaciones específicas para problemas de satisfacibilidad booleana

4.2. Arquitectura de la Ontología

La ontología se estructura siguiendo el modelo PEAS (Performance, Environment, Actuators, Sensors) para agentes inteligentes, proporcionando un framework completo para la respuesta automatizada a incidentes de ciberseguridad.

4.2.1. Definición de Tipos Básicos *(Sorts)*

La ontología define cinco tipos fundamentales utilizando la función `DeclareSort` de *Z3*:

```
ThreatSort = DeclareSort("Threat")      # Amenazas de seguridad
AssetSort = DeclareSort("Asset")        # Activos organizacionales
AttackSort = DeclareSort("Attack")      # Tipos de ataques especificos
UserSort = DeclareSort("User")          # Usuarios del sistema
IncidentSort = DeclareSort("Incident")  # Incidentes de seguridad
```

Listing 1: Definición de Tipos Básicos en Z3

Estos sorts actúan como los tipos fundamentales sobre los cuales se construyen todas las relaciones y predicados de la ontología.

4.2.2. Predicados y Funciones Ontológicas

La ontología define un conjunto de predicados y funciones que capturan las relaciones semánticas del dominio de ciberseguridad:
Predicados de Relación:

- `affects(Attack, Asset, Bool)`: Define qué ataques afectan a qué activos
- `has_vulnerability(Asset, Bool)`: Indica si un activo tiene vulnerabilidades conocidas
- `is_compromised(Asset, Bool)`: Especifica el estado de compromiso de un activo
- `user_has_access(User, Asset, Bool)`: Define permisos de acceso legítimos

Funciones de Valoración:

- `threat_level(Attack, Int)`: Asigna un nivel numérico de amenaza (1–4)
- `asset_criticality(Asset, Int)`: Define la criticidad del activo (1–5)
- `requires_immediate_response(Incident, Bool)`: Determina urgencia de respuesta

4.2.3. Métricas PEAS

Para operacionalizar el modelo PEAS, se definen métricas específicas que permiten evaluar el rendimiento del agente:

```
response_time = Function("response_time", IncidentSort, IntSort())
containment_effectiveness = Function("containment_effectiveness",
    IncidentSort, IntSort())
false_positive = Function("false_positive", IncidentSort, BoolSort())
```

Listing 2: Definición de Métricas PEAS en Z3

4.3. Reglas de Inferencia

El núcleo de la ontología consiste en un conjunto de reglas lógicas que definen el comportamiento del sistema de respuesta automática. Estas reglas utilizan cuantificadores universales (`ForAll`) e implicaciones lógicas (`Implies`) para capturar el conocimiento experto del dominio.

4.3.1. Regla de Respuesta Crítica

```
ForAll([threat, asset, incident],
    Implies(
        And(has_vulnerability(asset),
            affects(threat, asset),
            threat_level(threat) >= 3),
        requires_immediate_response(incident)
    )
)
```

Listing 3: Regla de Respuesta Crítica en Z3

Esta regla establece que cualquier amenaza de nivel alto ($\geq$3) que afecte a un activo vulnerable requiere respuesta inmediata.

4.3.2. Regla de Protección de Activos Críticos

```
ForAll([asset, incident],
    Implies(
        And(is_compromised(asset),
            asset_criticality(asset) >= 4),
        requires_immediate_response(incident)
    )
)
```

Listing 4: Regla de Protección de Activos Críticos en Z3

Especifica que el compromiso de activos de alta criticidad ($\geq$4) siempre requiere respuesta inmediata, independientemente del tipo de amenaza.

4.3.3. Regla de Tiempo de Respuesta

```
ForAll([incident],
    Implies(
        requires_immediate_response(incident),
        response_time(incident) <= 15
    )
)
```

Listing 5: Regla de Tiempo de Respuesta en Z3

Establece el constraint temporal de que los incidentes críticos deben ser atendidos en un máximo de 15 minutos.

4.4. Implementación del Modelo PEAS

4.4.1. Sensores *(Sensors)*

Se implementaron tres tipos de sensores especializados que alimentan datos a la ontología: Sensor de Anomalías de Red (`CybersecuritySensors.network_anomaly_sensor`):

- Detecta patrones de tráfico inusuales

- Genera automáticamente entidades de amenaza en Z3
- Asigna niveles de amenaza basados en la severidad detectada

Sensor de Integridad de Archivos (`CybersecuritySensors.file_integrity_sensor`):

- Monitorea cambios no autorizados en archivos críticos
- Marca activos como comprometidos en la base de conocimiento
- Vincula cambios de archivos con activos específicos

Sensor de Comportamiento de Usuario (`CybersecuritySensors.user_behavior_sensor`):

- Implementa análisis UEBA (User and Entity Behavior Analytics)
- Detecta accesos anómalos mediante análisis de patrones
- Actualiza predicados de acceso en tiempo real

4.4.2. Actuadores *(Actuators)*

Los actuadores implementan las respuestas automáticas del sistema:
Bloqueo de Tráfico Malicioso (`CyberSecurityActuators.block_malicious_traffic`):

- Implementa medidas de contención de red
- Registra acciones tomadas para auditoría
- Actualiza el estado del sistema en la ontología

Aislamiento de Dispositivos (`CyberSecurityActuators.isolate_compromised_device`):

- Ejecuta protocolos de cuarentena automatizada
- Modifica predicados de compromiso en Z3
- Mantiene trazabilidad de dispositivos aislados

Generación de Reportes (`CyberSecurityActuators.generate_incident_report`):

- Produce documentación estructurada de incidentes
- Integra análisis de Z3 con métricas operacionales
- Facilita el cumplimiento regulatorio y auditorías

4.4.3. Motor de Razonamiento *(Reasoning Engine)*

El componente central (`CyberSecurityReasoningEngine`) integra todos los elementos de la ontología:
Análisis de Amenazas:

- Crea instancias locales del solver Z3 para cada incidente
- Aplica hechos específicos basados en datos de sensores
- Evalúa reglas de inferencia para determinar respuestas

Evaluación de Severidad:

- Utiliza el modelo Z3 para calcular niveles de amenaza
- Implementa lógica de fallback para casos edge
- Mapea valores numéricos a categorías semánticas

4.4.4. Ventajas del Enfoque Adoptado

La construcción de la ontología mediante Z3 SMT Solver proporciona ventajas significativas sobre enfoques tradicionales:

- Verificación Formal: Garantías matemáticas de consistencia
- Razonamiento Eficiente: Optimizaciones específicas para satisfacibilidad

- Integración Directa: Sin overhead de translation entre lenguajes
- Expresividad: Capacidad de manejar lógica de primer orden compleja
- Extensibilidad: Fácil adición de nuevas reglas y predicados

Referencias

Gruber, T. (2009). Ontology [Entry in Encyclopedia of Database Systems]. En L. Liu & M. T. Özsu (Eds.), *Encyclopedia of Database Systems*. Springer-Verlag. https://tomgruber.org/writing/definition-of-ontology.pdf

W3C OWL Working Group. (2012, diciembre). *OWL 2 Web Ontology Language Document Overview (Second Edition)* (W3C Recommendation) (Editors: W3C OWL Working Group. This document provides a high-level overview of OWL 2 and its related specifications.). World Wide Web Consortium (W3C). https://www.w3.org/TR/owl2-overview/

Knublauch, H., & Kontokostas, D. (2017a, junio). *Shapes Constraint Language (SHACL)* (W3C Recommendation) (Editors: RDF Data Shapes Working Group. This document defines SHACL, a language for validating RDF graphs against a set of conditions.). World Wide Web Consortium (W3C). https://www.w3.org/TR/shacl/

RDF Working Group. (2014, febrero). *Resource Description Framework (RDF)* (W3C Recommendation) (RDF is a standard model for data interchange on the Web, supporting schema evolution and data merging across applications.). World Wide Web Consortium (W3C). https://www.w3.org/RDF/?utm_source=chatgpt.com

Baader, F. (2003). Basic Description Logics [Lecture notes from the Description Logic Handbook course, Free University of Bozen-Bolzano].

Knublauch, H., & Kontokostas, D. (2017b, junio). *Shapes Constraint Language (SHACL)* (W3C Recommendation) (W3C Recommendation published by the RDF Data Shapes Working Group). World Wide Web Consortium (W3C). https://www.w3.org/TR/2017/REC-shacl-20170720/

Martín, A., Celestino, S., Valdenebro, A., & Mensaque, J. (2020). Ontologías e Inteligencia Artificial para la Recuperación Eficiente del Conocimiento. *XV Jornadas Bibliotecarias de Andalucía*. https://es.scribd.com/document/55436704/10-Ontologia-e-Inteligencia-Artificial

A. Código Fuente

```
# -*- coding: utf-8 -*-
# %%
# type: ignore

import enum
from typing import Any, Dict, List, Optional, Tuple

from z3.z3 import (
    And,
    BoolSort,
    BoolVal,
    CheckSatResult,
    Const,
    DeclareSort,
    ExprRef,
    ForAll,
    Function,
    Implies,
    IntSort,
    ModelRef,
    Not,
    Solver,
    sat,
)

# %%


class ThreatLevel(enum.Enum):
    LOW = 1
    MEDIUM = 2
    HIGH = 3
```

```
    CRITICAL = 4


class AssetType(enum.Enum):
    SERVER = 1
    NETWORK_DEVICE = 2
    IOT_DEVICE = 3
    WORKSTATION = 4
    CLOUD_SERVICE = 5


class AttackType(enum.Enum):
    MALWARE = 1
    PHISHING = 2
    DDoS = 3
    DATA_BREACH = 4
    RANSOMWARE = 5
    INSIDER_THREAT = 6


# %%

ThreatSort = DeclareSort("Threat")
AssetSort = DeclareSort("Asset")
AttackSort = DeclareSort("Attack")
UserSort = DeclareSort("User")
IncidentSort = DeclareSort("Incident")

# %%
# functions and predicates for the ontology

affects = Function("affects", AttackSort, AssetSort, BoolSort())
has_vulnerability = Function("has_vulnerability", AssetSort, BoolSort())
threat_level = Function("threat_level", AttackSort, IntSort())
```

```
asset_criticality = Function("asset_criticality", AssetSort, IntSort())
requires_immediate_response = Function(
    "requires_immediate_response", IncidentSort, BoolSort()
)
user_has_access = Function("user_has_access", UserSort, AssetSort,
   BoolSort())
is_compromised = Function("is_compromised", AssetSort, BoolSort())

# %%
# PEAS metrics

response_time = Function("response_time", IncidentSort, IntSort())
containment_effectiveness = Function(
    "containment_effectiveness", IncidentSort, IntSort()
)
false_positive = Function("false_positive", IncidentSort, BoolSort())

# %%
solver = Solver()

# %%
threat = Const("threat", AttackSort)
asset = Const("asset", AssetSort)
user = Const("user", UserSort)
incident = Const("incident", IncidentSort)

# %%
# type: ignore

solver.add(
    ForAll(
        [threat, asset, incident],
        Implies(
            And(
```

```
                has_vulnerability(asset),
                affects(threat, asset),
                threat_level(threat) >= 3,
            ),
            requires_immediate_response(incident),
        ),
    )
)

solver.add(
    ForAll(
        [asset, incident],
        Implies(
            And(is_compromised(asset), asset_criticality(asset) >= 4),
            requires_immediate_response(incident),
        ),
    )
)

solver.add(
    ForAll(
        [user, asset],
        Implies(
            And(Not(user_has_access(user, asset)), is_compromised(asset)),
            BoolVal(True),
        ),
    )
)

solver.add(
    ForAll(
        [incident],
        Implies(
```

```
            requires_immediate_response(incident), response_time(incident)
    <= 15
        ),
    )
)

# %%
# type: ignore


class CybersecuritySensors:
    def __init__(self, solver: Solver) -> None:
        self.solver: Solver = solver

    def network_anomaly_sensor(
        self, traffic_data: Dict[str, Any]
    ) -> Optional[ExprRef]:
        anomaly_detected: bool = traffic_data.get("unusual_patterns",
   False)

        if anomaly_detected:
            new_threat: ExprRef = Const(
                f"network_threat_{id(traffic_data)}", AttackSort
            )
            self.solver.add(threat_level(new_threat) >= 2)
            return new_threat

        return None

    def file_integrity_sensor(
        self, file_changes: Dict[str, Any]
    ) -> Optional[ExprRef]:
        """Monitor de integridad de archivos"""
        if file_changes.get("unauthorized_changes", False):
```

```
            affected_asset: ExprRef = Const(
                f"asset_{file_changes['asset_id']}", AssetSort
            )
            self.solver.add(is_compromised(affected_asset))
            return affected_asset

        return None

    def user_behavior_sensor(
        self, user_activity: Dict[str, Any]
    ) -> Optional[Tuple[ExprRef, ExprRef]]:
        """Sensor UEBA (User and Entity Behavior Analytics)"""
        if user_activity.get("anomalous_behavior", False):
            suspicious_user: ExprRef = Const(
                f"user_{user_activity['user_id']}", UserSort
            )
            target_asset: ExprRef = Const(
                f"asset_{user_activity['target_asset']}", AssetSort
            )
            self.solver.add(Not(user_has_access(suspicious_user,
    target_asset)))
            return (suspicious_user, target_asset)

        return None


# %%
# type: ignore


class CyberSecurityActuators:
    def __init__(self, solver: Solver) -> None:
        self.solver: Solver = solver

```

```
    def block_malicious_traffic(self, threat_source: str) -> bool:
        return True

    def isolate_compromised_device(self, asset_id: str) -> bool:
        """Aislamiento de dispositivos comprometidos"""
        print(f"Aislando dispositivo comprometido: {asset_id}")
        isolated_asset: ExprRef = Const(f"asset_{asset_id}", AssetSort)
        self.solver.add(Not(is_compromised(isolated_asset)))
        return True

    def generate_incident_report(
        self, incident_data: Dict[str, Any]
    ) -> Dict[str, Any]:
        """Generar reporte detallado del incidente"""
        analysis: Dict[str, Any] = incident_data.get("analysis", {})

        report: Dict[str, Any] = {
            "timestamp": incident_data.get("timestamp"),
            "severity": incident_data.get("severity"),
            "affected_assets": incident_data.get("assets", []),
            "recommended_actions": analysis.get("recommendations", []),
            "immediate_response_triggered": analysis.get(
                "immediate_response", False
            ),
            "actions_taken": incident_data.get("actions_taken", []),
            "z3_reasoning_used": analysis.get("z3_model_used", False),
        }

        return report

    def _get_recommendations(self, incident_data: Dict[str, Any]) -> List[
    str]:
        return ["Aplicar parches", "Monitorear actividad", "Revisar logs"]

```

```python

# %%
# type: ignore


class CyberSecurityReasoningEngine:
    def __init__(
        self,
        solver: Solver,
        sensors: CybersecuritySensors,
        actuators: CyberSecurityActuators,
    ) -> None:
        self.solver: Solver = solver
        self.sensors: CybersecuritySensors = sensors
        self.actuators: CyberSecurityActuators = actuators

    def analyze_threat(self, threat_data: Dict[str, Any]) -> Dict[str, Any]:
        local_solver: Solver = Solver()

        for assertion in self.solver.assertions():
            local_solver.add(assertion)

        current_threat: ExprRef = Const(f"threat_{id(threat_data)}", AttackSort)
        current_asset: ExprRef = Const(f"asset_{id(threat_data)}", AssetSort)
        current_incident: ExprRef = Const(
            f"incident_{id(threat_data)}", IncidentSort
        )

        # Agregar hechos basados en los datos del incidente
        self._add_threat_facts(
            local_solver,
```

```
            current_threat,
            current_asset,
            current_incident,
            threat_data,
        )

        # Verificar si el modelo es satisfacible        check_result: CheckSatResult = local_solver.check()
        if check_result == sat:
            model: ModelRef = local_solver.model()

            # Evaluar usando el modelo Z3
            threat_severity: ThreatLevel = self._evaluate_threat_severity_z3(
                current_threat, model, threat_data
            )
            response_needed: bool = self._requires_immediate_response_z3(
                current_incident, model, threat_data
            )
            recommendations: List[str] = self._generate_recommendations_z3(
                model, threat_data, current_threat, current_asset
            )

            return {
                "severity": threat_severity,
                "immediate_response": response_needed,
                "recommendations": recommendations,
                "z3_model_used": True,
            }
        else:
            return {"error": "Inconsistencia en la ontologia detectada"}

    def _add_threat_facts(
```

```
        self,
        local_solver: Solver,
        threat: ExprRef,
        asset: ExprRef,
        incident: ExprRef,
        threat_data: Dict[str, Any],
    ) -> None:
        if threat_data.get("affects_critical_assets", False):
            local_solver.add(threat_level(threat) == 4)  # CRITICAL
        elif threat_data.get("widespread_impact", False):
            local_solver.add(threat_level(threat) == 3)  # HIGH
        elif threat_data.get("network_anomaly", False):
            local_solver.add(threat_level(threat) == 2)  # MEDIUM
        else:
            local_solver.add(threat_level(threat) == 1)  # LOW

        if threat_data.get("affects_critical_assets", False):
            local_solver.add(asset_criticality(asset) == 5)  # Maxima
criticidad
        else:
            local_solver.add(asset_criticality(asset) == 2)  # Criticidad
normal

        # Establecer relaciones
        local_solver.add(affects(threat, asset))

        if threat_data.get("compromised_assets", []):
            local_solver.add(is_compromised(asset))

        if threat_data.get("vulnerability_detected", False):
            local_solver.add(has_vulnerability(asset))

    def _evaluate_threat_severity_z3(
```

```
        self, threat: ExprRef, model: ModelRef, threat_data: Dict[str, Any]
    ) -> ThreatLevel:
        """Evaluar severidad usando el modelo Z3"""
        try:
            # Intentar evaluar el nivel de amenaza desde el modelo
            threat_level_val: Optional[ExprRef] = model.evaluate(
                threat_level(threat), model_completion=True
            )

            if threat_level_val is not None:
                level: int = threat_level_val.as_long()
                if level >= 4:
                    return ThreatLevel.CRITICAL
                elif level >= 3:
                    return ThreatLevel.HIGH
                elif level >= 2:
                    return ThreatLevel.MEDIUM
                else:
                    return ThreatLevel.LOW
        except Exception as e:
            print(f"Error evaluando nivel de amenaza: {e}")

        if threat_data.get("affects_critical_assets", False):
            return ThreatLevel.CRITICAL
        elif threat_data.get("widespread_impact", False):
            return ThreatLevel.HIGH
        else:
            return ThreatLevel.MEDIUM

    def _requires_immediate_response_z3(
        self, incident: ExprRef, model: ModelRef, threat_data: Dict[str, Any]
    ) -> bool:
```

```
356         """Determinar respuesta inmediata usando Z3"""
357         try:
358             immediate_response: Optional[ExprRef] = model.evaluate(
359                 requires_immediate_response(incident), model_completion=
    True
360             )
361 
362             if immediate_response is not None:
363                 return bool(immediate_response)
364         except Exception as e:
365             print(f"Error evaluando respuesta inmediata: {e}")
366 
367         # Fallback
368         return threat_data.get(
369             "affects_critical_assets", False
370         ) or threat_data.get("active_exploitation", False)
371 
372     def _generate_recommendations_z3(
373         self,
374         model: ModelRef,
375         threat_data: Dict[str, Any],
376         threat: ExprRef,
377         asset: ExprRef,
378     ) -> List[str]:
379         recommendations: List[str] = []
380 
381         try:
382             # Evaluar diferentes aspectos del modelo
383             is_asset_compromised: Optional[ExprRef] = model.evaluate(
384                 is_compromised(asset), model_completion=True
385             )
386             threat_level_val: Optional[ExprRef] = model.evaluate(
387                 threat_level(threat), model_completion=True
388             )
```

```
            asset_critical: Optional[ExprRef] = model.evaluate(
                asset_criticality(asset), model_completion=True
            )

            # Recomendaciones basadas en el estado del activo
            if is_asset_compromised and bool(is_asset_compromised):
                recommendations.append(
                    "CRITICO: Aislar activo comprometido inmediatamente"
                )
                recommendations.append(
                    "Realizar analisis forense del activo afectado"
                )

            # Recomendaciones basadas en nivel de amenaza
            if threat_level_val and threat_level_val.as_long() >= 3:
                recommendations.append(
                    "Activar protocolo de respuesta de emergencia"
                )
                recommendations.append("Notificar al equipo directivo")

            # Recomendaciones basadas en criticidad del activo
            if asset_critical and asset_critical.as_long() >= 4:
                recommendations.append("Implementar monitoreo continuo
    24/7")
                recommendations.append(
                    "Realizar backup inmediato de datos criticos"
                )

        except Exception as e:
            print(f"Error evaluando modelo Z3: {e}")

        # Recomendaciones adicionales basadas en tipo de amenaza
        if threat_data.get("network_anomaly", False):
```

```
421             recommendations.append("Analizar trafico de red en tiempo real
    ")
422             recommendations.append(
423                 "Implementar reglas de firewall restrictivas"
424             )
425
426         if threat_data.get("user_anomaly", False):
427             recommendations.append(
428                 "Revisar credenciales y permisos de usuario"
429             )
430             recommendations.append("Forzar cambio de contrase\u00f1as")
431
432         if threat_data.get("malware_detected", False):
433             recommendations.append(
434                 "Ejecutar escaneo completo de antimalware"
435             )
436             recommendations.append(
437                 "Limpiar y desinfectar sistemas afectados"
438             )
439
440         if not recommendations:
441             recommendations = [
442                 "Documentar incidente en el sistema SIEM",
443                 "Incrementar nivel de monitoreo",
444                 "Revisar logs de seguridad",
445             ]
446
447         return recommendations
448
449     def respond_to_incident(
450         self, incident_data: Dict[str, Any]
451     ) -> Dict[str, Any]:
452         print(
```

```
453             f"Analizando incidente: {incident_data.get('timestamp', 'N/A')
    }"
454         )
455
456         analysis: Dict[str, Any] = self.analyze_threat(incident_data)
457
458         if "error" in analysis:
459             return analysis
460
461         severity: ThreatLevel = analysis["severity"]
462         print(f"[SEVERIDAD] Severidad detectada: {severity.name}")
463         print(
464             f"[ALERTA] Respuesta inmediata requerida: {analysis['
    immediate_response']}"
465         )
466
467         # Ejecutar actuadores si es necesario
468         actions_taken: List[str] = []
469         if analysis.get("immediate_response", False):
470             if incident_data.get("network_threat"):
471                 source: str = incident_data.get("source", "unknown")
472                 if self.actuators.block_malicious_traffic(source):
473                     actions_taken.append(f"Bloqueado trafico desde {source
    }")
474
475             compromised_assets: List[str] = incident_data.get(
476                 "compromised_assets", []
477             )
478             if compromised_assets:
479                 for asset in compromised_assets:
480                     if self.actuators.isolate_compromised_device(asset):
481                         actions_taken.append(f"Aislado dispositivo {asset}
    ")
482
```

```
        # Generar reporte detallado
        report: Dict[str, Any] = self.actuators.generate_incident_report(
            {
                "timestamp": incident_data.get("timestamp"),
                "severity": analysis.get("severity"),
                "assets": incident_data.get("compromised_assets", []),
                "analysis": analysis,
                "actions_taken": actions_taken,
            }
        )

        report["actions_taken"] = actions_taken
        report["z3_reasoning"] = analysis.get("z3_model_used", False)

        return report


# %%
sensors = CybersecuritySensors(solver=solver)
actuators = CyberSecurityActuators(solver=solver)
reasoning_engine = CyberSecurityReasoningEngine(
    solver=solver, sensors=sensors, actuators=actuators
)
```