



Universidad Tecnológica de Bolívar

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

23 de septiembre de 2025

Índice

1	Introduccion	3
1.1	Caso de Estudio	3
1.2	Resumen Ejecutivo	3
2	Modulo <i>PEAS</i>	3
2.1	Performance (Medición del Desempeño)	4
2.2	Environment (Entorno)	4
2.3	Actuators (Actuadores)	4
2.4	Sensors (Sensores)	5
3	Ontología	5
3.1	Definicion	5
3.2	¿De qué está hecha una ontología?	5
3.3	Lenguajes y estándares que la hacen operativa	6
3.4	Usos clave	6
3.5	Ontología en la Inteligencia Artificial	7
4	Construcción de la Ontología	8
4.1	Metodología de Desarrollo	8
4.2	Arquitectura de la Ontología	8
4.2.1	Definición de Tipos Básicos (<i>Sorts</i>)	9
4.2.2	Predicados y Funciones Ontológicas	9
4.2.3	Métricas PEAS	10
4.3	Reglas de Inferencia	10
4.3.1	Regla de Respuesta Crítica	10
4.3.2	Regla de Protección de Activos Críticos	11
4.3.3	Regla de Tiempo de Respuesta	11
4.4	Implementación del Modelo PEAS	11

4.4.1	Sensores (<i>Sensors</i>)	11
4.4.2	Actuadores (<i>Actuators</i>)	12
4.4.3	Motor de Razonamiento (<i>Reasoning Engine</i>)	13
4.4.4	Ventajas del Enfoque Adoptado	13
Conclusiones		14
Recomendaciones		14
Referencias		16
Nota sobre la representacion de la ontologia		17
A Código Fuente		18

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. Definición

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*:

```
1 ThreatSort = DeclareSort("Threat")      # Amenazas de seguridad
2 AssetSort = DeclareSort("Asset")        # Activos organizacionales
3 AttackSort = DeclareSort("Attack")      # Tipos de ataques específicos
4 UserSort = DeclareSort("User")          # Usuarios del sistema
5 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:

```
1 response_time = Function("response_time", IncidentSort, IntSort())
2 containment_effectiveness = Function("containment_effectiveness",
    IncidentSort, IntSort())
3 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

```
1 ForAll([threat, asset, incident],
2     Implies(
3         And(has_vulnerability(asset),
4             affects(threat, asset),
5             threat_level(threat) >= 3),
6         requires_immediate_response(incident)
7     )
8 )
```

Listing 3: Regla de Respuesta Crítica en Z3

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

4.3.2. Regla de Protección de Activos Críticos

```
1 ForAll([asset, incident],
2     Implies(
3         And(is_compromised(asset),
4             asset_criticality(asset) >= 4),
5         requires_immediate_response(incident)
6     )
7 )
```

Listing 4: Regla de Protección de Activos Críticos en Z3

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

4.3.3. Regla de Tiempo de Respuesta

```
1 ForAll([incident],
2     Implies(
3         requires_immediate_response(incident),
4         response_time(incident) <= 15
5     )
6 )
```

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

Conclusiones

El desarrollo e implementación de un agente inteligente de ciberseguridad permitió comprender la importancia de modelar formalmente el conocimiento mediante ontologías, así como la utilidad de herramientas de razonamiento lógico como Z3 para la verificación y análisis automático de relaciones y restricciones en el dominio. A lo largo del trabajo, se evidenció que la representación estructurada y formal de conceptos facilita la detección de amenazas, la toma de decisiones automatizada y la adaptabilidad del sistema ante nuevos escenarios.

Si bien la ontología no se encuentra en un formato estándar como OWL o TTL, la aproximación lógica utilizada demostró ser efectiva para el razonamiento y la validación de reglas complejas. Este enfoque resalta la flexibilidad de la inteligencia artificial para abordar problemas de seguridad desde distintas perspectivas técnicas.

Como posibles mejoras futuras, se sugiere la integración de la ontología en plataformas interoperables, la incorporación de aprendizaje automático para enriquecer la detección de amenazas y la expansión del modelo a otros dominios de la ciberseguridad. En conclusión, el uso de agentes inteligentes y ontologías formales representa una estrategia robusta y escalable para enfrentar los desafíos actuales en la protección de sistemas digitales.

Recomendaciones

A partir del desarrollo realizado, se proponen las siguientes recomendaciones y líneas de trabajo futuro:

- **Interoperabilidad semántica:** Traducir la ontología desarrollada en Z3 a formatos estándar como OWL o TTL, permitiendo su integración con plataformas de la Web Semántica y herramientas como Protégé.
- **Integración de aprendizaje automático:** Incorporar técnicas de machine learning para enriquecer la detección de amenazas y permitir que el agente evolucione y se adapte a nuevos patrones de ataque de manera autónoma.
- **Validación en entornos reales:** Probar el agente en escenarios de ciberseguridad reales o simulados, evaluando su desempeño, robustez y capacidad de respuesta ante amenazas complejas.
- **Expansión del dominio:** Ampliar la ontología y el agente para cubrir otros ámbitos de la ciberseguridad, como la protección de infraestructuras críticas, IoT o sistemas industriales.
- **Interfaz de usuario y visualización:** Desarrollar interfaces gráficas que permitan a los usuarios interactuar con el agente, visualizar el estado del sistema y comprender las decisiones tomadas por el modelo.
- **Colaboración multiagente:** Explorar la posibilidad de implementar sistemas multi-agente, donde varios agentes colaboren o compitan para mejorar la defensa cibernética de manera distribuida.
- **Consideraciones éticas y legales:** Analizar los aspectos éticos y regulatorios asociados al uso de agentes inteligentes en ciberseguridad, especialmente en lo referente a privacidad y toma de decisiones automatizada.

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>

Nota sobre la representacion de la ontologia

Disclaimer: La ontología presentada en este trabajo fue implementada utilizando el solucionador lógico Z3, lo que permite realizar razonamiento formal y verificación automática sobre los conceptos y relaciones definidos. Sin embargo, no se generó un archivo en formato OWL (.owl) o Turtle (.ttl), ya que el enfoque adoptado priorizó la expresividad lógica y la capacidad de inferencia computacional sobre la interoperabilidad con herramientas de la Web Semántica. En caso de requerirse una representación en OWL o TTL, sería necesario traducir manualmente los conceptos, relaciones y restricciones aquí modelados a dichos formatos utilizando un editor de ontologías.

A. Código Fuente

```
1 # -*- coding: utf-8 -*-
2 # %%
3 # type: ignore
4
5 import enum
6 from typing import Any, Dict, List, Optional, Tuple
7
8 from z3.z3 import (
9     And,
10    BoolSort,
11    BoolVal,
12    CheckSatResult,
13    Const,
14    DeclareSort,
15    ExprRef,
16    ForAll,
17    Function,
18    Implies,
19    IntSort,
20    ModelRef,
21    Not,
22    Solver,
23    sat,
24 )
25
26 # %%
27
28
29 class ThreatLevel(enum.Enum):
30     LOW = 1
31     MEDIUM = 2
32     HIGH = 3
```

```

33     CRITICAL = 4
34
35
36 class AssetType(enum.Enum):
37     SERVER = 1
38     NETWORK_DEVICE = 2
39     IOT_DEVICE = 3
40     WORKSTATION = 4
41     CLOUD_SERVICE = 5
42
43
44 class AttackType(enum.Enum):
45     MALWARE = 1
46     PHISHING = 2
47     DDoS = 3
48     DATA_BREACH = 4
49     RANSOMWARE = 5
50     INSIDER_THREAT = 6
51
52
53 # %%
54
55 ThreatSort = DeclareSort("Threat")
56 AssetSort = DeclareSort("Asset")
57 AttackSort = DeclareSort("Attack")
58 UserSort = DeclareSort("User")
59 IncidentSort = DeclareSort("Incident")
60
61 # %%
62 # functions and predicates for the ontology
63
64 affects = Function("affects", AttackSort, AssetSort, BoolSort())
65 has_vulnerability = Function("has_vulnerability", AssetSort, BoolSort())
66 threat_level = Function("threat_level", AttackSort, IntSort())

```

```

67 asset_criticality = Function("asset_criticality", AssetSort, IntSort())
68 requires_immediate_response = Function(
69     "requires_immediate_response", IncidentSort, BoolSort()
70 )
71 user_has_access = Function("user_has_access", UserSort, AssetSort,
72     BoolSort())
73
74 # %%
75 # PEAS metrics
76
77 response_time = Function("response_time", IncidentSort, IntSort())
78 containment_effectiveness = Function(
79     "containment_effectiveness", IncidentSort, IntSort()
80 )
81 false_positive = Function("false_positive", IncidentSort, BoolSort())
82
83 # %%
84 solver = Solver()
85
86 # %%
87 threat = Const("threat", AttackSort)
88 asset = Const("asset", AssetSort)
89 user = Const("user", UserSort)
90 incident = Const("incident", IncidentSort)
91
92 # %%
93 # type: ignore
94
95 solver.add(
96     ForAll(
97         [threat, asset, incident],
98         Implies(
99             And(

```

```

100         has_vulnerability(asset),
101         affects(threat, asset),
102         threat_level(threat) >= 3,
103     ),
104     requires_immediate_response(incident),
105 ),
106 )
107 )
108
109 solver.add(
110     ForAll(
111         [asset, incident],
112         Implies(
113             And(is_compromised(asset), asset_criticality(asset) >= 4),
114             requires_immediate_response(incident),
115         ),
116     )
117 )
118
119 solver.add(
120     ForAll(
121         [user, asset],
122         Implies(
123             And(Not(user_has_access(user, asset)), is_compromised(asset)),
124             BoolVal(True),
125         ),
126     )
127 )
128
129 solver.add(
130     ForAll(
131         [incident],
132         Implies(

```

```

133         requires_immediate_response(incident), response_time(incident)
134         <= 15
135     ),
136 )
137
138 # %%
139 # type: ignore
140
141
142 class CybersecuritySensors:
143     def __init__(self, solver: Solver) -> None:
144         self.solver: Solver = solver
145
146     def network_anomaly_sensor(
147         self, traffic_data: Dict[str, Any]
148     ) -> Optional[ExprRef]:
149         anomaly_detected: bool = traffic_data.get("unusual_patterns",
150 False)
151
152         if anomaly_detected:
153             new_threat: ExprRef = Const(
154                 f"network_threat_{id(traffic_data)}", AttackSort
155             )
156             self.solver.add(threat_level(new_threat) >= 2)
157             return new_threat
158
159         return None
160
161     def file_integrity_sensor(
162         self, file_changes: Dict[str, Any]
163     ) -> Optional[ExprRef]:
164         """Monitor de integridad de archivos"""
165         if file_changes.get("unauthorized_changes", False):

```

```

165         affected_asset: ExprRef = Const(
166             f"asset_{file_changes['asset_id']}", AssetSort
167         )
168         self.solver.add(is_compromised(affected_asset))
169         return affected_asset
170
171     return None
172
173     def user_behavior_sensor(
174         self, user_activity: Dict[str, Any]
175     ) -> Optional[Tuple[ExprRef, ExprRef]]:
176         """Sensor UEBA (User and Entity Behavior Analytics)"""
177         if user_activity.get("anomalous_behavior", False):
178             suspicious_user: ExprRef = Const(
179                 f"user_{user_activity['user_id']}", UserSort
180             )
181             target_asset: ExprRef = Const(
182                 f"asset_{user_activity['target_asset']}", AssetSort
183             )
184             self.solver.add(Not(user_has_access(suspicious_user,
185 target_asset)))
186
187             return (suspicious_user, target_asset)
188
189         return None
190
191     # %%
192     # type: ignore
193
194     class CyberSecurityActuators:
195         def __init__(self, solver: Solver) -> None:
196             self.solver: Solver = solver
197

```



```

198 def block_malicious_traffic(self, threat_source: str) -> bool:
199     return True
200
201 def isolate_compromised_device(self, asset_id: str) -> bool:
202     """Aislamiento de dispositivos comprometidos"""
203     print(f"Aislando dispositivo comprometido: {asset_id}")
204     isolated_asset: ExprRef = Const(f"asset_{asset_id}", AssetSort)
205     self.solver.add(Not(is_compromised(isolated_asset)))
206     return True
207
208 def generate_incident_report(
209     self, incident_data: Dict[str, Any]
210 ) -> Dict[str, Any]:
211     """Generar reporte detallado del incidente"""
212     analysis: Dict[str, Any] = incident_data.get("analysis", {})
213
214     report: Dict[str, Any] = {
215         "timestamp": incident_data.get("timestamp"),
216         "severity": incident_data.get("severity"),
217         "affected_assets": incident_data.get("assets", []),
218         "recommended_actions": analysis.get("recommendations", []),
219         "immediate_response_triggered": analysis.get(
220             "immediate_response", False
221         ),
222         "actions_taken": incident_data.get("actions_taken", []),
223         "z3_reasoning_used": analysis.get("z3_model_used", False),
224     }
225
226     return report
227
228 def _get_recommendations(self, incident_data: Dict[str, Any]) -> List[
229     str]:
230     return ["Aplicar parches", "Monitorear actividad", "Revisar logs"]

```

```

231
232 # %%
233 # type: ignore
234
235
236 class CyberSecurityReasoningEngine:
237     def __init__(
238         self,
239         solver: Solver,
240         sensors: CybersecuritySensors,
241         actuators: CyberSecurityActuators,
242     ) -> None:
243         self.solver: Solver = solver
244         self.sensors: CybersecuritySensors = sensors
245         self.actuators: CyberSecurityActuators = actuators
246
247     def analyze_threat(self, threat_data: Dict[str, Any]) -> Dict[str, Any
248 ]:
249
250         local_solver: Solver = Solver()
251
252         for assertion in self.solver.assertions():
253             local_solver.add(assertion)
254
255         current_threat: ExprRef = Const(f"threat_{id(threat_data)}",
256 AttackSort)
257
258         current_asset: ExprRef = Const(f"asset_{id(threat_data)}",
259 AssetSort)
260
261         current_incident: ExprRef = Const(
262             f"incident_{id(threat_data)}", IncidentSort
263         )
264
265         # Agregar hechos basados en los datos del incidente
266         self._add_threat_facts(
267             local_solver,

```

```

262         current_threat,
263         current_asset,
264         current_incident,
265         threat_data,
266     )
267
268     # Verificar si el modelo es satisfacible           check_result:
269     CheckSatResult = local_solver.check()
270     if check_result == sat:
271         model: ModelRef = local_solver.model()
272
273         # Evaluar usando el modelo Z3
274         threat_severity: ThreatLevel = self.
275         _evaluate_threat_severity_z3(
276             current_threat, model, threat_data
277         )
278         response_needed: bool = self._requires_immediate_response_z3(
279             current_incident, model, threat_data
280         )
281         recommendations: List[str] = self._generate_recommendations_z3
282         (
283             model, threat_data, current_threat, current_asset
284         )
285
286         return {
287             "severity": threat_severity,
288             "immediate_response": response_needed,
289             "recommendations": recommendations,
290             "z3_model_used": True,
291         }
292     else:
293         return {"error": "Inconsistencia en la ontologia detectada"}
294
295     def _add_threat_facts(

```

```

293     self,
294     local_solver: Solver,
295     threat: ExprRef,
296     asset: ExprRef,
297     incident: ExprRef,
298     threat_data: Dict[str, Any],
299 ) -> None:
300     if threat_data.get("affects_critical_assets", False):
301         local_solver.add(threat_level(threat) == 4) # CRITICAL
302     elif threat_data.get("widespread_impact", False):
303         local_solver.add(threat_level(threat) == 3) # HIGH
304     elif threat_data.get("network_anomaly", False):
305         local_solver.add(threat_level(threat) == 2) # MEDIUM
306     else:
307         local_solver.add(threat_level(threat) == 1) # LOW
308
309     if threat_data.get("affects_critical_assets", False):
310         local_solver.add(asset_criticality(asset) == 5) # Maxima
311         criticidad
312     else:
313         local_solver.add(asset_criticality(asset) == 2) # Criticidad
314         normal
315
316     # Establecer relaciones
317     local_solver.add(affects(threat, asset))
318
319     if threat_data.get("compromised_assets", []):
320         local_solver.add(is_compromised(asset))
321
322     if threat_data.get("vulnerability_detected", False):
323         local_solver.add(has_vulnerability(asset))
324
325 def _evaluate_threat_severity_z3(

```

```

324     self, threat: ExprRef, model: ModelRef, threat_data: Dict[str, Any
    ]
325 ) -> ThreatLevel:
326     """Evaluar severidad usando el modelo Z3"""
327     try:
328         # Intentar evaluar el nivel de amenaza desde el modelo
329         threat_level_val: Optional[ExprRef] = model.evaluate(
330             threat_level(threat), model_completion=True
331         )
332
333         if threat_level_val is not None:
334             level: int = threat_level_val.as_long()
335             if level >= 4:
336                 return ThreatLevel.CRITICAL
337             elif level >= 3:
338                 return ThreatLevel.HIGH
339             elif level >= 2:
340                 return ThreatLevel.MEDIUM
341             else:
342                 return ThreatLevel.LOW
343     except Exception as e:
344         print(f"Error evaluando nivel de amenaza: {e}")
345
346     if threat_data.get("affects_critical_assets", False):
347         return ThreatLevel.CRITICAL
348     elif threat_data.get("widespread_impact", False):
349         return ThreatLevel.HIGH
350     else:
351         return ThreatLevel.MEDIUM
352
353     def _requires_immediate_response_z3(
354         self, incident: ExprRef, model: ModelRef, threat_data: Dict[str,
Any]
355     ) -> 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         )

```

```

389         asset_critical: Optional[ExprRef] = model.evaluate(
390             asset_criticality(asset), model_completion=True
391         )
392
393     # Recomendaciones basadas en el estado del activo
394     if is_asset_compromised and bool(is_asset_compromised):
395         recommendations.append(
396             "CRITICO: Aislar activo comprometido inmediatamente"
397         )
398         recommendations.append(
399             "Realizar analisis forense del activo afectado"
400         )
401
402     # Recomendaciones basadas en nivel de amenaza
403     if threat_level_val and threat_level_val.as_long() >= 3:
404         recommendations.append(
405             "Activar protocolo de respuesta de emergencia"
406         )
407         recommendations.append("Notificar al equipo directivo")
408
409     # Recomendaciones basadas en criticidad del activo
410     if asset_critical and asset_critical.as_long() >= 4:
411         recommendations.append("Implementar monitoreo continuo
24/7")
412         recommendations.append(
413             "Realizar backup inmediato de datos criticos"
414         )
415
416     except Exception as e:
417         print(f"Error evaluando modelo Z3: {e}")
418
419     # Recomendaciones adicionales basadas en tipo de amenaza
420     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

```

```

483     # Generar reporte detallado
484     report: Dict[str, Any] = self.actuators.generate_incident_report(
485         {
486             "timestamp": incident_data.get("timestamp"),
487             "severity": analysis.get("severity"),
488             "assets": incident_data.get("compromised_assets", []),
489             "analysis": analysis,
490             "actions_taken": actions_taken,
491         }
492     )
493
494     report["actions_taken"] = actions_taken
495     report["z3_reasoning"] = analysis.get("z3_model_used", False)
496
497     return report
498
499
500 # %%
501 sensors = CybersecuritySensors(solver=solver)
502 actuators = CyberSecurityActuators(solver=solver)
503 reasoning_engine = CyberSecurityReasoningEngine(
504     solver=solver, sensors=sensors, actuators=actuators
505 )

```