CyberSecOnto - Un ontologia per la Cybersecurity

Web Reasoning - Progetto Finale

SALVATORE LEANZA

Catania, Anno Accademico 2025/2026

Università di Catania

Docente:
Prof. Marianna Nicolosi Asmundo

Indice

Capitolo 1

Introduzione

La sicurezza informatica moderna (Cybersecurity) è caratterizzata da una vasta eterogeneità di informazioni: asset hardware, vulnerabilità software, tipologie di attacco e contromisure. La gestione di queste informazioni tramite database relazionali tradizionali risulta spesso rigida e incapace di evidenziare relazioni implicite, come l'impatto a catena di un guasto o il livello di rischio derivato.

In questo progetto è stata sviluppata CyberSecOnto, un'ontologia in linguaggio OWL (*Web Ontology Language*) progettata per modellare lo scenario della sicurezza aziendale.

1.1 Obiettivi del progetto

L'obiettivo principale non è la semplice archiviazione dei dati, ma l'automazione del processo di analisi del rischio. Attraverso l'uso di tecnologie del Web Semantico, il sistema è in grado di:

- **Modellare** l'infrastruttura IT e le dipendenze tra i sistemi.
- **Inferire automaticamente** la criticità delle vulnerabilità basandosi su punteggi standard (CVSS).
- **Classificare** i sistemi a rischio senza intervento umano, sfruttando sia regole SWRL che inferenze OWL standard.
- **Validare** la consistenza dei dati tramite restrizioni logiche.

Capitolo 2

Modellazione Concettuale

La struttura terminologica (T-Box) è stata definita utilizzando l'editor **Protégé**. Il dominio è stato modellato attraverso una gerarchia di classi, proprietà semantiche e restrizioni logiche.

2.1 Gerarchia delle Classi

Le entità principali seguono una tassonomia che riflette le componenti di un ecosistema cyber:

- **System**: La classe radice per gli asset hardware. Include le sottoclassi `Server` e `Workstation`.
- **Vulnerability**: Rappresenta le debolezze software (es. CVE). Include la sottoclasse `SoftwareVulnerability`.
- **Attack**: Rappresenta le tipologie di attacco (es. Malware, Ransomware).
- **Mitigation**: Rappresenta le contromisure difensive (es. Firewall, Patch).
- **ThreatActor**: Gli agenti di minaccia (es. Hacker, NationState).

2.1.1 Assiomi di Disgiunzione

Per garantire la coerenza logica del modello, è stata applicata la disgiunzione (*Disjoint Classes*) tra le classi di alto livello. Ad esempio, un individuo non può appartenere contemporaneamente alla classe `System` e alla classe `AttackType`.

2.2 Proprietà

Le relazioni semantiche definiscono le interazioni tra gli individui.

2.2.1 Object Properties

- `hasVulnerability` (Domain: System → Range: Vulnerability): Associa un asset alla sua debolezza.
- `exploits` (Domain: AttackType → Range: Vulnerability): Indica quale falla viene sfruttata da un attacco.
- `mitigates` (Domain: Mitigation → Range: Vulnerability): Indica quale difesa risolve il problema.
- `dependsOn` (Domain: System → Range: System): Modella la dipendenza operativa tra macchine.

La Transitività La proprietà `dependsOn` è stata definita come **Transitiva**. Se il Sistema A dipende da B, e B dipende da C, il *Reasoner* inferisce automaticamente che A dipende da C. Questo è fondamentale per l'analisi dell'impatto a catena in infrastrutture complesse.

2.2.2 Data Properties

- `hasCVSSScore` (Domain: Vulnerability → Range: `xsd:decimal`): Assegna il punteggio numerico di gravità (Common Vulnerability Scoring System). È stato scelto il tipo `decimal` per garantire la compatibilità con i calcoli di precisione.

2.3 Restrizioni Locali

Per aumentare la robustezza e l'espressività del modello, sono state implementate tre tipologie di restrizioni OWL.

2.3.1 Restrizione di Cardinalità (Exactly)

Per garantire l'integrità dei dati, è stato imposto che ogni vulnerabilità debba avere un unico punteggio di gravità.

- **Classe:** `Vulnerability`
- **Assioma:** `hasCVSSScore exactly 1 xsd:decimal`

Inoltre, la proprietà `hasCVSSScore` è stata resa **Funzionale**, generando un'inconsistenza (errore) qualora a un individuo vengano assegnati due punteggi distinti.

2.3.2 Restrizione Universale (Only)

Per vincolare il dominio degli attacchi, è stato stabilito che un attacco può sfruttare *soltanto* vulnerabilità.

- **Classe:** `AttackType`
- **Assioma:** `exploits only Vulnerability`

Questo previene errori di modellazione, come il collegamento errato di un attacco direttamente a un server.

2.3.3 Restrizione Esistenziale (Some)

È stata definita una classe speciale per la classificazione automatica.

- **Classe:** `VulnerableSystem`
- **Assioma:** `System and (hasVulnerability some Vulnerability)`

Capitolo 3

Reasoning e Inferenza

Il valore aggiunto del progetto risiede nella capacità di dedurre nuova conoscenza non esplicitamente asserita. Sono state utilizzate due tecniche complementari: le regole **SWRL** e l'inferenza standard **OWL**.

3.1 Regole SWRL (Semantic Web Rule Language)

Le regole SWRL sono state utilizzate per calcoli che coinvolgono valori numerici e propagazione condizionale.

3.1.1 Regola 1: Classificazione Vulnerabilità Critica

Questa regola valuta il punteggio CVSS. Se supera la soglia di 8.0, la vulnerabilità viene riclassificata.

```
Vulnerability(?v) ^ hasCVSSScore(?v, ?score) ^ swrlb:greaterThan(?
    score, 8.0)
 -> CriticalVulnerability(?v)
```

Listing 3.1: Regola CriticalVulnerability

3.1.2 Regola 2: Propagazione del Rischio (High Risk)

Questa regola propaga la criticità dalla vulnerabilità al sistema che la ospita.

```
System(?s) ^ hasVulnerability(?s, ?v) ^ CriticalVulnerability(?v)
 -> HighRiskSystem(?s)
```

Listing 3.2: Regola HighRiskSystem

3.2 Inferenza tramite Classi Equivalenti (OWL)

Per rispondere alla necessità di classificare i sistemi in base alla presenza di vulnerabilità (indipendentemente dalla gravità), è stata utilizzata una **Classe Equivalente** definita tramite restrizione esistenziale.

La classe `VulnerableSystem` è stata definita come:

```
System and (hasVulnerability some Vulnerability)
```

Il *Reasoner* (Pellet) analizza gli individui definiti come generici `System`: se rileva la presenza di almeno una relazione `hasVulnerability`, inferisce automaticamente l'appartenenza alla classe `VulnerableSystem`.

Capitolo 4

Implementazione Software e Automazione

Per gestire la complessità e garantire la scalabilità del progetto, lo sviluppo non si è limitato all'uso dell'editor grafico Protégé. È stata implementata una pipeline di automazione completa per simulare scenari Enterprise e superare le limitazioni computazionali dei reasoner standard.

4.1 Ambiente di Sviluppo (Visual Studio Code)

L'intero ciclo di vita del software è stato gestito utilizzando **Visual Studio Code**. Il progetto è stato strutturato in un repository **GitHub** che contenente:

- I file sorgente dell'ontologia (`Untitled.rdf`, `cyberseconto.owl`).
- Gli script Python di automazione (`popolamento.py`, `apply_swrl_rules.py`).
- Gli script di test e validazione (`test_sparql.py`, `debug_onto.py`).

L'uso di un sistema di controllo versione ha permesso di tracciare l'evoluzione del modello semantico e di separare la logica applicativa dai dati ontologici.

4.2 Integrazione Python con Owlready2

L'interazione programmatica con l'ontologia è stata realizzata tramite la libreria **Owlready2**, che consente di mappare le classi OWL in classi Python. Questo ha permesso di manipolare individui e proprietà come oggetti nativi del linguaggio.

4.3 Script di Generazione Procedurale

Lo script `popolamento.py` funge da simulatore di infrastruttura. Invece di popolare manualmente, lo script:

1. Carica il file base (`Untitled.rdf`).

2. Istanzia proceduralmente 20 `Server` con nomi univoci.

3. Genera 10 vulnerabilità (CVE) con punteggi CVSS randomizzati (distribuzione 4.0 - 10.0).

4. Crea istanze di `Attack` e le collega casualmente alle vulnerabilità.

5. Salva il risultato in `cyberseconto_populated.owl`.

4.4 Motore Inferenziale Ibrido

Data la complessità computazionale di alcuni Reasoner nel gestire atomi SWRL matematici su dataset estesi, è stato implementato un motore di inferenza esterno tramite lo script `apply_swrl_rules.py`.

Questo script analizza i dati, applica la logica delle regole SWRL e inietta i risultati direttamente nell'ontologia. Questo approccio permette di generare un file finale denominato `cyberseconto_inferred.owl`, che rappresenta la versione definitiva, più densa e completa dell'intero progetto. Tale file racchiude la totalità della base di conoscenza, consolidando in un unico ambiente sia i dati popolati che le deduzioni logiche estratte, rendendole persistenti e indipendenti dal software di visualizzazione utilizzato.

Capitolo 5

Interrogazione (SPARQL)

Le query SPARQL presentate in questo capitolo sono state il frutto di un processo iterativo di progettazione. Inizialmente, le interrogazioni sono state pensate e generate programmaticamente all'interno degli script Python per facilitare l'estrazione automatizzata dei dati durante la fase di sviluppo. Successivamente, tali query sono state validate e testate singolarmente nell'ambiente Protégé per assicurarne la correttezza semantica e la precisione dei risultati. È importante sottolineare che nella presente relazione sono state riportate esclusivamente alcune tra le query ai fini della dimostrazione dell'efficacia del modello. L'intera suite di interrogazioni, comprese quelle utilizzate per il debugging e la manutenzione dei dati, è consultabile integralmente all'interno degli script di repository

5.1 Query: Report Sistemi Critici

Questa query estrae tutti i sistemi che il Reasoner ha classificato come "Ad Alto Rischio" e mostra il relativo punteggio di gravità, ordinandoli in modo decrescente.

```
PREFIX : <http://www.semanticweb.org/salvleanza/ontologies/
    cybersOnto#>
SELECT ?system ?vuln ?score
WHERE {
    ?system a :HighRiskSystem .
    ?system :hasVulnerability ?vuln .
    ?vuln :hasCVSSScore ?score .
}
ORDER BY DESC(?score)
```

Listing 5.1: SPARQL Query per HighRiskSystem

5.2 Query: Sistemi Vulnerabili (Inferenza OWL)

Questa query interroga la classe definita tramite restrizione esistenziale.

```
PREFIX : <http://www.semanticweb.org/salvleanza/ontologies/
    cybersOnto#>
SELECT ?system
WHERE {
    ?system a :VulnerableSystem .
}
```

Listing 5.2: SPARQL Query per VulnerableSystem

Capitolo 6

Conclusioni

Il progetto **CyberSecOnto** ha dimostrato l'efficacia delle ontologie nella cybersecurity, trasformando inventari statici in sistemi di analisi proattiva. L'integrazione tra OWL, SWRL e Python ha permesso di ottenere un modello scalabile e persistente, i cui sorgenti completi e script di automazione sono disponibili sulla repository GitHub dedicata e dettagliata: `https://github.com/salvlea/Semantic_Web.git`.