**DL expressivity enhancements in the aftermath of the Case Study:**

The DL expressivity concurrent with examination of the Case Study was estimated by manual means to be ALHI(O)

* AL (Attributive Language with complement)
* H (Role hierarchy)
* I (Inverse properties)
* (O) (Nominals)

In the post-Case Study period we undertook a series of enhancements to the ontology, specifically focusing on enhancing its DL expressivity, which is now ALCHFRSIOQ(D). The outcome of this effort is documented below.

**DL expressivity enhancement to the ontology: -- C**

**‘** Disjoint Classes/mutually exclusive concepts ‘

**Results of query:**

\*\*\*

=== Disjoint Classes ===

------------------------------

StateActor is disjoint with NonStateActor

NonStateActor is disjoint with StateActor

**Turtle representation from the ontology:**

**\*\*\***

fght:StateActor a owl:Class ;

    rdfs:subClassOf fght:ThreatActor ;

    rdfs:label "StateActor" ;

    rdfs:comment "Threat actors sponsored by or affiliated with nation states" .

fght:NonStateActor a owl:Class ;

    rdfs:subClassOf fght:ThreatActor ;

    rdfs:label "NonStateActor" ;

    rdfs:comment "Independent threat actors not affiliated with nation states" .

# Add directly after the StateActor and NonStateActor class definitions

[ rdf:type owl:AllDisjointClasses ;

  owl:members ( fght:StateActor fght:NonStateActor )

] .

**SPARQL query producing results (embedded in Python code):**

\*\*\*

from rdflib import Graph

g = Graph()

g.parse(r"C:\Users\andre\ontology\Decoding5G\_test\Ontology\_framework\_Jan21.ttl", format="turtle")

disjoint\_query = """

PREFIX fght: <https://purl.org/5g-hybrid-threats#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?class1 ?class2

WHERE {

   ?disj rdf:type owl:AllDisjointClasses ;

         owl:members ?list .

   ?list rdf:rest\*/rdf:first ?class1 .

   ?list rdf:rest\*/rdf:first ?class2 .

   FILTER(?class1 != ?class2)

}

"""

print("\n=== Disjoint Classes ===")

print("-" \* 30)

for row in g.query(disjoint\_query):

   class1 = str(row.class1).split('#')[1]

   class2 = str(row.class2).split('#')[1]

   print(f"{class1} is disjoint with {class2}")

**DL expressivity enhancement to the ontology: ‘ Collaboration/shared tools ’ --** R (complex role inclusion/property chains)

**Results of query:**

\*\*\*

Comprehensive APT Collaboration Analysis:

---------------------------------------

Campaign: SonyPicturesHack2014

Shared Tool: Mimikatz

APT28 Risk Profile:

- Risk Level: HighRisk

- Likelihood: HighLikelihood

- Severity: HighSeverity

LazarusGroup Risk Profile:

- Risk Level: HighRisk

- Likelihood: HighLikelihood

- Severity: Not specified

--------------------------------------------------

Campaign: SonyPicturesHack2014

Shared Tool: Tor Browser

APT28 Risk Profile:

- Risk Level: HighRisk

- Likelihood: HighLikelihood

- Severity: HighSeverity

LazarusGroup Risk Profile:

- Risk Level: HighRisk

- Likelihood: HighLikelihood

- Severity: Not specified

--------------------------------------------------

**Turtle representation from the ontology:**

**\*\*\***

# Sony Campaign Instance

fght:SonyPicturesHack2014 rdf:type fght:Campaign ;

    fght:targets fght:Vulnerability ; # Link campaigns to vulnerabilities - Jan 21 addition

    rdfs:label "Sony Pictures Hack 2014" ;

    rdfs:comment "Joint cyber operation involving data theft and leaks from Sony Pictures" ;

    fght:year "2014"^^xsd:gYear ;

    fght:hasParticipant fght:LazarusGroup, fght:APT28 ;

    fght:attributedTo fght:SolarWinds\_Attack ;

    fght:source "CrowdStrike" .

**SPARQL query producing results (embedded in Python code):**

from rdflib import Graph

g = Graph()

g.parse(r"C:\Users\andre\ontology\Decoding5G\_test\Ontology\_framework\_Jan21.ttl", format="turtle")

final\_query = """

PREFIX fght: <https://purl.org/5g-hybrid-threats#>

SELECT DISTINCT ?campaign ?malwareName ?risk1 ?risk2 ?likelihood1 ?likelihood2 ?severity1 ?severity2

WHERE {

    # Campaign and shared tools

    ?campaign fght:hasParticipant fght:APT28 ;

             fght:hasParticipant fght:LazarusGroup .

    ?malware fght:MalwareName ?malwareName ;

            fght:usedBy fght:APT28 ;

            fght:usedBy fght:LazarusGroup .

    # Risk profiles

    OPTIONAL { fght:APT28 fght:hasRiskLevel ?risk1 }

    OPTIONAL { fght:APT28 fght:hasRiskLikelihood ?likelihood1 }

    OPTIONAL { fght:APT28 fght:hasRiskSeverity ?severity1 }

    OPTIONAL { fght:LazarusGroup fght:hasRiskLevel ?risk2 }

    OPTIONAL { fght:LazarusGroup fght:hasRiskLikelihood ?likelihood2 }

    OPTIONAL { fght:LazarusGroup fght:hasRiskSeverity ?severity2 }

}

"""

print("\nComprehensive APT Collaboration Analysis:")

print("---------------------------------------")

for row in g.query(final\_query):

    campaign = str(row.campaign).split('#')[1]

    tool = str(row.malwareName)

    risk1 = str(row.risk1).split('#')[1] if row.risk1 else "Not specified"

    risk2 = str(row.risk2).split('#')[1] if row.risk2 else "Not specified"

    likelihood1 = str(row.likelihood1).split('#')[1] if row.likelihood1 else "Not specified"

    likelihood2 = str(row.likelihood2).split('#')[1] if row.likelihood2 else "Not specified"

    severity1 = str(row.severity1).split('#')[1] if row.severity1 else "Not specified"

    severity2 = str(row.severity2).split('#')[1] if row.severity2 else "Not specified"

    print(f"\nCampaign: {campaign}")

    print(f"Shared Tool: {tool}")

    print("\nAPT28 Risk Profile:")

    print(f"- Risk Level: {risk1}")

    print(f"- Likelihood: {likelihood1}")

    print(f"- Severity: {severity1}")

    print("\nLazarusGroup Risk Profile:")

    print(f"- Risk Level: {risk2}")

    print(f"- Likelihood: {likelihood2}")

    print(f"- Severity: {severity2}")

    print("-" \* 50)

\*\*\*

**DL expressivity enhancement to the ontology: ‘Property Chains for complex relationships/partOfGroup’ – SIR**

DL expressivity features demonstrated are:

1. S (Transitivity) - through group membership relationships
2. I (Inverse roles) - shown by owl:inverseOf fght:hasParticipant
3. R (Complex role chains) - demonstrated by owl:propertyChainAxiom combining inverse participation relationships

This example demonstrates SIR expressivity features working together to infer collaborations through group memberships and inverse relationships.

**Results of query:**

\*\*\*

Inferred Collaborations through Group Membership:

----------------------------------------------

APT28 inferred to collaborate with APT42

**Turtle representation from the ontology:**

# Properties for Collaboration Chain

fght:hasParticipant rdf:type owl:ObjectProperty ;

    rdfs:domain fght:Campaign ;

    rdfs:range fght:ThreatActor ;

    rdfs:comment "Links a campaign to its participating threat actors" .

# Property Chain for Collaboration

fght:collaboratesWith owl:propertyChainAxiom (

    owl:inverseOf fght:hasParticipant

    fght:hasParticipant

) .

**SPARQL query producing results (embedded in Python code):**

\*\*\*

from rdflib import Graph

g = Graph()

g.parse(r"C:\Users\andre\ontology\Decoding5G\_test\Ontology\_framework\_Jan21.ttl", format="turtle")

inference\_query = """

PREFIX fght: <https://purl.org/5g-hybrid-threats#>

SELECT DISTINCT ?actor1 ?actor2

WHERE {

    # Check inferred collaborations through property chain

    ?actor1 fght:partOfGroup ?group1 .

    ?actor2 fght:partOfGroup ?group2 .

    ?actor1 fght:collaboratesWith ?actor2 .

    # Filter to ensure we're looking at different actors

    FILTER(?actor1 != ?actor2)

}

"""

print("\nInferred Collaborations through Group Membership:")

print("----------------------------------------------")

for row in g.query(inference\_query):

    actor1 = str(row.actor1).split('#')[1]

    actor2 = str(row.actor2).split('#')[1]

    print(f"{actor1} inferred to collaborate with {actor2}")

1. **DL expressivity enhancement to the ontology: ‘ subPartOf ’ -- S**

(Transitive Properties defining hierarchical relationships of sections of the ENISA 5G Threat Landscape Report December 2020)

\*\*\*

**Results of query:**

\*\*\*

Direct subPartOf relationships:

--------------------------------

VulnerabilityGroupsCoreNetwork subPartOf Section4\_5GVulnerabilities

Section4\_5GVulnerabilities subPartOf AnnexC\_DetailedVulnerabilities

All relationships (including inferred):

--------------------------------

VulnerabilityGroupsCoreNetwork subPartOf Section4\_5GVulnerabilities

VulnerabilityGroupsCoreNetwork subPartOf AnnexC\_DetailedVulnerabilities

Section4\_5GVulnerabilities subPartOf AnnexC\_DetailedVulnerabilities

**Turtle representation from the ontology:**

**\*\*\***

# Document Section Class

fght:DocumentSection rdf:type owl:Class ;

    rdfs:comment "Represents a section or part of a document" .

# SubPartOf Property

fght:subPartOf rdf:type owl:ObjectProperty, owl:TransitiveProperty ;

    rdfs:domain fght:DocumentSection ;

    rdfs:range fght:DocumentSection ;

    rdfs:comment "Indicates that one document section is part of another section" .

# Specific Sections

fght:VulnerabilityGroupsCoreNetwork rdf:type fght:DocumentSection ;

    rdfs:label "VULNERABILITY GROUPS FOR CORE NETWORK" .

fght:Section4\_5GVulnerabilities rdf:type fght:DocumentSection ;

    rdfs:label "Section 4. 5G VULNERABILITIES" .

fght:AnnexC\_DetailedVulnerabilities rdf:type fght:DocumentSection ;

    rdfs:label "C ANNEX: DETAILED VULNERABILITIES IN THE CORE NETWORK" .

# Relationships

fght:VulnerabilityGroupsCoreNetwork fght:subPartOf fght:Section4\_5GVulnerabilities .

fght:Section4\_5GVulnerabilities fght:subPartOf fght:AnnexC\_DetailedVulnerabilities .

**SPARQL query producing results (embedded in Python code):**

\*\*\*

from rdflib import Graph, Namespace

from rdflib.plugins.sparql import prepareQuery

# Create graph

g = Graph()

# Load your ontology file with correct path

g.parse(r"C:\Users\andre\ontology\Decoding5G\_test\Ontology\_framework\_Jan21.ttl", format="turtle")

# SPARQL query for direct relationships

direct\_query = """

PREFIX fght: <https://purl.org/5g-hybrid-threats#>

SELECT DISTINCT ?subject ?object

WHERE {

    ?subject fght:subPartOf ?object .

}

"""

# SPARQL query for transitive path

transitive\_query = """

PREFIX fght: <https://purl.org/5g-hybrid-threats#>

SELECT DISTINCT ?start ?end

WHERE {

    ?start fght:subPartOf+ ?end .

}

"""

print("\nDirect subPartOf relationships:")

print("--------------------------------")

for row in g.query(direct\_query):

    subj = str(row.subject).split('#')[1]

    obj = str(row.object).split('#')[1]

    print(f"{subj} subPartOf {obj}")

print("\nAll relationships (including inferred):")

print("--------------------------------")

for row in g.query(transitive\_query):

    subj = str(row.start).split('#')[1]

    obj = str(row.end).split('#')[1]

    print(f"{subj} subPartOf {obj}")

**DL expressivity enhancement to the ontology: ‘ part of group’ and ‘collaborates with’ and ‘cardinality’ -- Q, S, R**

The DL expressivity features are:

1. Q (Qualified cardinality restrictions) - shown by owl:minQualifiedCardinality "2" constraining group membership
2. S (Transitivity) - demonstrated in group membership hierarchies
3. R (Complex role chains) - shown by property chains inferring collaborations between groups

**Results of query:**

1. Group Memberships:

-------------------

APT28 is member of RussianAPTs

APT29 is member of RussianAPTs

APT41 is member of ChineseAPTs

APT42 is member of ChineseAPTs

ENISA\_ETL\_2020 is member of ENISA\_ETL\_Series

ENISA\_ETL\_Series is member of ENISA\_Publications

2. Inferred Collaborations through Property Chain:

----------------------------------------------

APT28 (in RussianAPTs) collaborates with APT42 (in ChineseAPTs)

**Turtle representation from the ontology:**

**\*\*\***

**Cardinality code from ontology:**

*i.e. CineseAPTs and RussianAPTs must have at least 2 members*

# Add hasMember property

fght:hasMember rdf:type owl:ObjectProperty ;

rdfs:domain fght:ThreatActorGroup ;

rdfs:range fght:ThreatActor ;

rdfs:comment "Links a threat group to its member actors" .

# Add cardinality constraint

fght:ThreatActorGroup rdfs:subClassOf [

rdf:type owl:Restriction ;

owl:onProperty fght:hasMember ;

owl:minQualifiedCardinality "2"^^xsd:nonNegativeInteger ;

owl:onClass fght:ThreatActor

] .

**SPARQL query producing results (embedded in Python code):**

from rdflib import Graph

g = Graph()

g.parse(r"C:\Users\andre\ontology\Decoding5G\_test\Ontology\_framework\_Jan21.ttl", format="turtle")

# Query 1: Check group memberships

group\_query = """

PREFIX fght: <https://purl.org/5g-hybrid-threats#>

SELECT DISTINCT ?actor ?group

WHERE {

    ?actor fght:partOfGroup ?group .

}

"""

# Query 2: Check inferred collaborations

collab\_query = """

PREFIX fght: <https://purl.org/5g-hybrid-threats#>

SELECT DISTINCT ?actor1 ?group1 ?actor2 ?group2

WHERE {

    ?actor1 fght:partOfGroup ?group1 .

    ?actor2 fght:partOfGroup ?group2 .

    ?actor1 fght:collaboratesWith ?actor2 .

    # Filter to ensure we're looking at different actors and groups

    FILTER(?actor1 != ?actor2 && ?group1 != ?group2)

}

"""

print("\n1. Group Memberships:")

print("-------------------")

for row in g.query(group\_query):

    actor = str(row.actor).split('#')[1]

    group = str(row.group).split('#')[1]

    print(f"{actor} is member of {group}")

print("\n2. Inferred Collaborations through Property Chain:")

print("----------------------------------------------")

for row in g.query(collab\_query):

    actor1 = str(row.actor1).split('#')[1]

    group1 = str(row.group1).split('#')[1]

    actor2 = str(row.actor2).split('#')[1]

    group2 = str(row.group2).split('#')[1]

    print(f"{actor1} (in {group1}) collaborates with {actor2} (in {group2})")

1. **DL expressivity enhancement to the ontology: ‘Complex Class Constructors/unionOf’ -- C**

(Complex concept definition using boolean constructors)

This is demonstrated by:

1. Use of owl:unionOf to combine:
   * StateActor
   * Complex NonStateActor definition
2. owl:intersectionOf for NonStateActors with high threat
3. Results showing both types:
   * StateActors (APT41, AdvancedPersistentThreat, LazarusGroup)
   * High-threat NonStateActor (ExampleIdeologicalActor)

CyberThreatActor is either:

A StateActor (like AdvancedPersistentThreat, APT41)

OR

A NonStateActor that has a High threat level

Note: NonStateActor has a sub-Class called: IdeologicallyMotivatedPerson, and its Instance is called ExampleIdeologicalActor.

**Results of query:**

Complete CyberThreatActor Union Results:

-------------------------------------

Actor: ExampleIdeologicalActor

Category: HighThreat NonStateActor

Threat Level: Not specified

----------------------------------------

Actor: APT41

Category: StateActor

Threat Level: High

----------------------------------------

Actor: AdvancedPersistentThreat

Category: StateActor

Threat Level: Not specified

----------------------------------------

Actor: LazarusGroup

Category: StateActor

Threat Level: High

**----------------------------------------**

**Turtle representation from the ontology:**

# Add CyberThreatActor class with union

fght:CyberThreatActor owl:equivalentClass [

    owl:unionOf (

        fght:StateActor

        [ rdf:type owl:Class ;

          owl:intersectionOf (

              fght:NonStateActor

              [ rdf:type owl:Restriction ;

                owl:onProperty fght:hasThreatLevel ;

                owl:hasValue fght:High

              ]

          )

        ]

    )

] .

**SPARQL query producing results (embedded in Python code):**

from rdflib import Graph

g = Graph()

g.parse(r"C:\Users\andre\ontology\Decoding5G\_test\Ontology\_framework\_Jan21.ttl", format="turtle")

union\_query = """

PREFIX fght: <https://purl.org/5g-hybrid-threats#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?actor ?category ?threatLevel

WHERE {

  {

    # State Actors like APT28, APT41, etc.

    ?actor rdfs:subClassOf fght:StateActor .

    OPTIONAL { ?actor fght:hasThreatLevel ?threatLevel }

    BIND("StateActor" AS ?category)

  }

  UNION

  {

    # High-threat NonStateActors (IdeologicallyMotivatedPerson)

    ?actor rdf:type fght:IdeologicallyMotivatedPerson ;

           fght:hasThreatLevel fght:High .

    BIND("HighThreat NonStateActor" AS ?category)

  }

}

ORDER BY ?category ?actor

"""

print("\nComplete CyberThreatActor Union Results:")

print("-------------------------------------")

for row in g.query(union\_query):

    actor = str(row.actor).split('#')[1]

    category = str(row.category)

    level = str(row.threatLevel).split('#')[1] if row.threatLevel else "Not specified"

    print(f"Actor: {actor}")

    print(f"Category: {category}")

    print(f"Threat Level: {level}")

    print("-" \* 40)

\*\*\*

**\*\*\***

1. **DL expressivity enhancement to the ontology: ‘StateActors/threat levels/group memberships (partOfGroup)’ -- SFH**

The DL expressivity features demonstrated here are:

1. S (Transitivity) - group membership hierarchy
2. F (Functionality) - hasThreatLevel is functional (single value per actor)
3. H (Role hierarchy) - group membership relationships

**Results of query:**

State Actors with Threat Levels and Groups:

-----------------------------------------

Actor: APT41

Threat Level: High

Group: ChineseAPTs

----------------------------------------

Actor: APT28

Threat Level: High

Group: RussianAPTs

----------------------------------------

Actor: APT42

Threat Level: High

Group: ChineseAPTs

----------------------------------------

Actor: LazarusGroup

Threat Level: High

Group: No group

----------------------------------------

**Turtle representation from the ontology:**

**\*\*\***

# Define ThreatLevel class and instances

fght:ThreatLevel rdf:type owl:Class .

# Define threat level property

fght:hasThreatLevel rdf:type owl:ObjectProperty ;

    rdfs:domain fght:ThreatActor ;

    rdfs:range fght:ThreatLevel .

# ThreatLevel Instance

fght:High rdf:type fght:ThreatLevel ;

    rdfs:label "High Threat Level" .

# Add threat levels for State Actors

fght:APT41 fght:hasThreatLevel fght:High .

fght:APT28 fght:hasThreatLevel fght:High .

fght:APT42 fght:hasThreatLevel fght:High .

fght:LazarusGroup fght:hasThreatLevel fght:High .

**SPARQL query producing results (embedded in Python code):**

from rdflib import Graph

g = Graph()

g.parse(r"C:\Users\andre\ontology\Decoding5G\_test\Ontology\_framework\_Jan21.ttl", format="turtle")

state\_actor\_query = """

PREFIX fght: <https://purl.org/5g-hybrid-threats#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?actor ?threat ?group

WHERE {

    # First pattern: actors with threat levels

    {

        ?actor fght:hasThreatLevel ?threat

    }

    # Second pattern: group memberships

    OPTIONAL {

        ?actor fght:partOfGroup ?group

    }

}

"""

print("\nState Actors with Threat Levels and Groups:")

print("-----------------------------------------")

for row in g.query(state\_actor\_query):

    actor = str(row.actor).split('#')[1]

    threat = str(row.threat).split('#')[1] if row.threat else "No threat level"

    group = str(row.group).split('#')[1] if row.group else "No group"

    print(f"Actor: {actor}")

    print(f"Threat Level: {threat}")

    print(f"Group: {group}")

    print("-" \* 40)