MUNI

Graph-Based CPE Matching for Identification of Vulnerable Asset Configurations

2nd International Workshop on Graph-based Network Security

Daniel Tovarňák, Lukáš Sadlek, Pavel Čeleda

Masaryk University, CSIRT-MU

Brno, Czech Republic

May 21st, 2021

Vulnerabilities

Threat

- Foreseen or unforeseen **eventuality** with the potential to negatively impact security (of assets).

Weakness

- Shortcoming, error, or omission in procedures, standards, rules, policies, organization, system, network, hardware, or software, which allows for a threat to be realized under **favorable conditions**.

Vulnerability

- **Weakness** that truly allows for a threat to be realized, i.e. it is an **exploitable weakness**.

Assets

Asset

- Anything **valuable** to the organization
  - e.g. hardware, software, services, information, people, reputation
- We focus on HW and SW **assets** and their **components**
  - e.g. NICs, servers, operating systems, applications, libraries

Asset Configuration Examples

- Component Trees
  Firefox 43.0, running on top of Windows 10 v2004 en_US, on top of Intel Xeon processor, inside a Dell Laptop of a particular type.
- SW Dependency Trees
  Python & PyPI, Java & Maven, Javascript & npm, Debian & APT

Automation Support for HW and SW Vulnerabilities

Common Vulnerabilities and Exposures (CVE)

- De-facto standard for unambiguous identification and description of vulnerabilities in computer systems

Common Platform Enumeration (CPE)

- Naming standard for identifying classes of **applications (libraries), OS, hardware**, i.e. individual **asset components**
- Naming, Name matching, Dictionary, Applicability language

National Vulnerability Database (NVD)

- U.S. government repository containing **CVE vulnerabilities**
- Vulnerabilities are linked to vulnerable **asset configurations**

NVD CVE Record in YAML

```yaml
cve:
  CVE_data_meta: {ASSIGNER: cve@mitre.org, ID: CVE-2017-5754}
  # ... Meltdown
  description: { ... }
  problemtype: { ... } # CWE - Common Weakness Enumeration
  references: { ... }
impact: { ... } # CVSS - Common Vulnerability Scoring System
configurations: # CPE applicability statements
  CVE_data_version: 4.0
  nodes:
  - cpe_match:
    operator: OR
    # ...
    - cpe23Uri: "cpe:2.3:h:intel:atom_c:c2308:*:*:*:*:*:*:*"
      vulnerable: true
    - cpe23Uri: "cpe:2.3:h:intel:atom_c:c2316:*:*:*:*:*:*:*"
      vulnerable: true
    - cpe23Uri: "cpe:2.3:h:intel:xeon_silver:4116t:*:*:*:*:*:*:*"
      vulnerable: true
lastModifiedDate: 2020-05-05T11:31Z
publishedDate: 2018-01-04T13:29Z
```

Asset Inventory in YAML

```yaml
- name: windows-10-common
  uuid: a84fa9de-de42-443f-87b2-0d892ecdb8f5
  parentUuid: null
  cpe23: "cpe:2.3:o:microsoft:windows_10:1709:*:*:*:*:*:x64:*"

- name: firefox-browser-80.0
  uuid: 6f66140f-4979-41a7-b067-73860dbddc39
  parentUuid: null
  cpe23: "cpe:2.3:a:mozilla:firefox:80.0:*:*:*:*:android:*:*"

- name: node-tar-3.1.2
  uuid: 131b6b1c-3ba9-11eb-adc1-0242ac120002
  parentUuid: null
  cpe23: "cpe:2.3:a:node-tar_project:node-tar:3.1.2:*:*:*:*:*:*:*"

- name: node-tar-2.1.0
  uuid: 1311111c-0000-11eb-adc1-5555ac120002
  parentUuid: a84fa9de-de42-443f-87b2-0d892ecdb8f5
  cpe23: "cpe:2.3:a:node-tar_project:node-tar:2.1.0:*:*:*:*:*:*:*"
```

Vulnerability Management via CPE Matching

Motivation

- Determine if there are vulnerable assets in your inventory
- CVE list has more than **150,000 entries**
- Attackers exploit **even very old** vulnerabilities in HW and SW
- **Asset** component **configurations** are complex and ever-changing
- We want to track **every change** in asset inventory and CVE list
- Naïve approach is based on **exhaustive search**
- Graph-based approach may be more **efficient** (lower complexity)

CPE Naming

Well-Formed Name (WFN)

- Foundational logical data construct
- Unordered set of **attribute-value pairs**
- Bound or unbound form

Attribute-Value Pair

- Standard defines **11 attributes**
- Values are **exact** strings or **logical** values (ANY, NA)

```
wfn:[part="o", vendor="microsoft", product="windows_server_2008",
↪  version="r2", update="sp1", edition=ANY, language=ANY,
↪  sw_edition=ANY, target_sw=ANY, target_hw="itanium", other=ANY]
cpe:/o:microsoft:windows_server_2008:r2:sp1:~~~~itanium~
cpe:2.3:o:microsoft:windows_server_2008:r2:sp1:*:*:*:*:itanium:*
```

CPE Name Matching (Pairwise AV Comparison)

No.	Source AV pair	Target AV pair	AV-Relation
1	*ANY*	*ANY*	$=$
2	*ANY*	*NA*	$\supset$
3	*ANY*	i	$\supset$
4	*ANY*	m^+	undefined
5	*NA*	*ANY*	$\subset$
6	*NA*	*NA*	**=**
7	*NA*	i	$\neq$
8	*NA*	m^+	undefined
9	i	i	$=$
10	i	k	$\neq$
11	i	m^+	undefined
12	i	*NA*	$\neq$
13	i	*ANY*	$\subset$
14	m^+	i	$\supset$ or $\neq$
15	m^+	*ANY*	$\subset$
16	m^+	*NA*	$\neq$
17	m_1^+	m_2^+	undefined

CPE Name Matching

CPE Name Matching

- Two steps
  1. Pairwise **AV comparison** (result is a set of AV-Relations)
  2. **CPE name comparison** based on the result from the first step
- Possible results
  - If *any* AV-R is *DISJOINT* ($\neq$), the result is *DISJOINT*.
  - If *all* AV-Rs are *EQUAL* (=) or *SUPERSET* ($\supset$), the result is *SUPERSET*.
  - If *all* AV-Rs are *EQUAL* (=) or *SUBSET* ($\subset$), the result is *SUBSET*.

CPE Match

- Two CPEs are **matching**, if the **source CPE name** (CVE-side) is *SUPERSET* of the **target CPE name** (asset-side).

CPE Applicability Language

CPE Applicability Statements

- Each CVE record can define multiple *applicability statements*
  - *Chrome AND Windows 7* OR *Chrome v55.0*
- **Complex Boolean formulas** of bound CPE names (R_n^{CPE})

$$(R_1^{\text{CPE}} \wedge R_2^{\text{CPE}} \wedge \neg(R_3^{\text{CPE}} \vee R_4^{\text{CPE}})) \vee \neg R_5^{\text{CPE}} \tag{1}$$

Conjunctive Normal Form

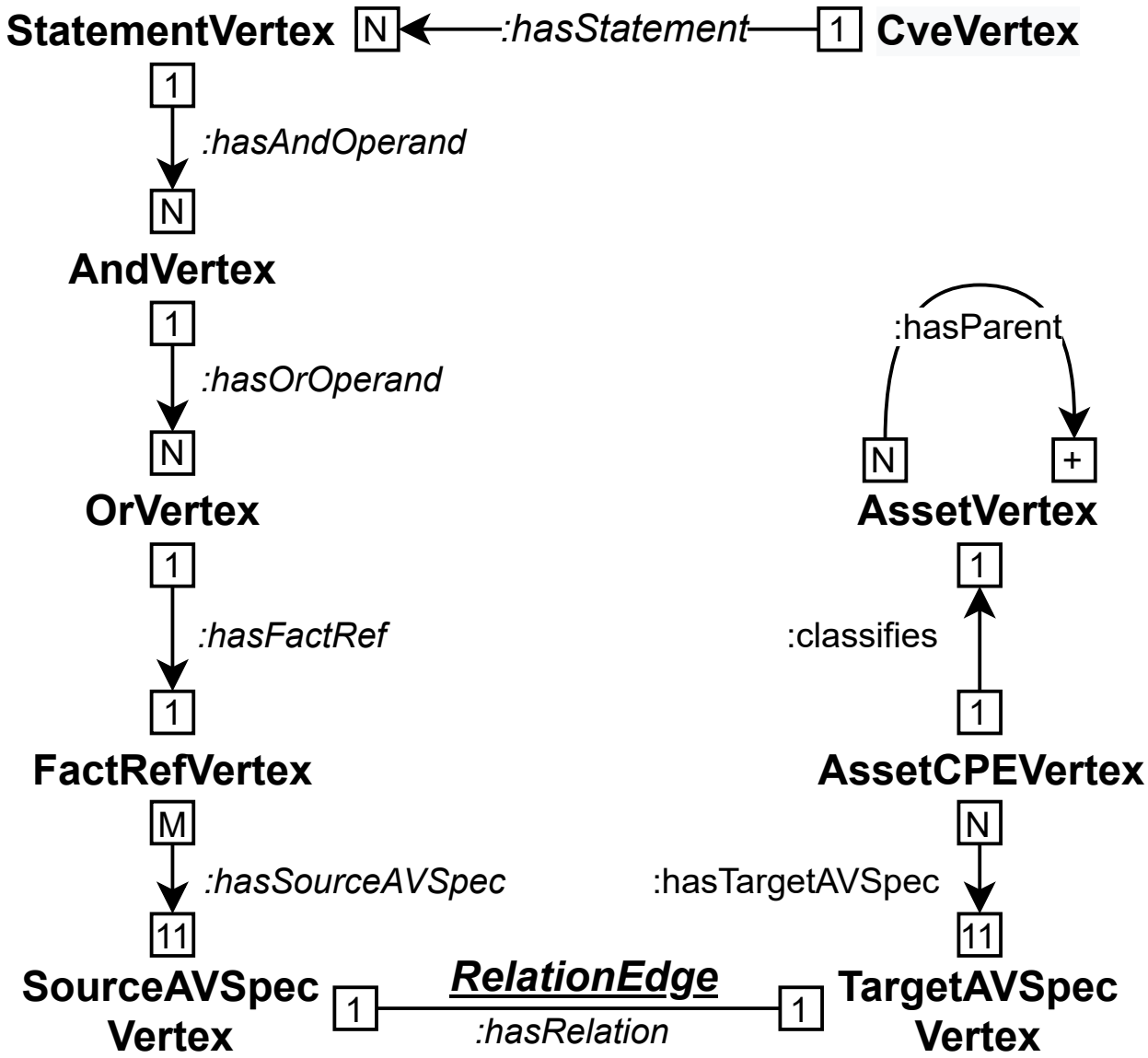
- *Conjunction* ($\wedge$) consisting of one or more *clauses*, each of which is a disjunction ($\vee$) of one or more literals.
- The logical complement ($\neg$) can be present only for literals.

$$(R_1^{\text{CPE}} \vee \neg R_5^{\text{CPE}}) \wedge (R_2^{\text{CPE}} \vee \neg R_5^{\text{CPE}}) \wedge (\neg R_3^{\text{CPE}} \vee \neg R_5^{\text{CPE}}) \wedge (\neg R_4^{\text{CPE}} \vee \neg R_5^{\text{CPE}}) \tag{2}$$

Proposed Graph-Based Approach

- Aims to eliminate **exhaustive search**
- **Graph model** for storing CVEs and their CPE applicability statements and assets with CPE definitions
  - Relies on **Conjunctive Normal Form**
- **Insertion procedure** that performs AV pairwise comparison
- **Graph search query** that finds all the vulnerable (`CveVertex`, `StatementVertex`, `AssetVertex`) tuples in a **single pass**

Graph Model


StatementVertex
N
:hasStatement
1
CveVertex
1
:hasAndOperand
N
AndVertex
1
:hasOrOperand
N
OrVertex
1
:hasFactRef
1
FactRefVertex
M
:hasSourceAVSpec
11
SourceAVSpec Vertex
:hasParent
N
+
AssetVertex
1
:classifies
1
AssetCPEVertex
N
:hasTargetAVSpec
11
TargetAVSpec Vertex
1
RelationEdge
:hasRelation
1

Insertion Procedure

Insert CVE Record

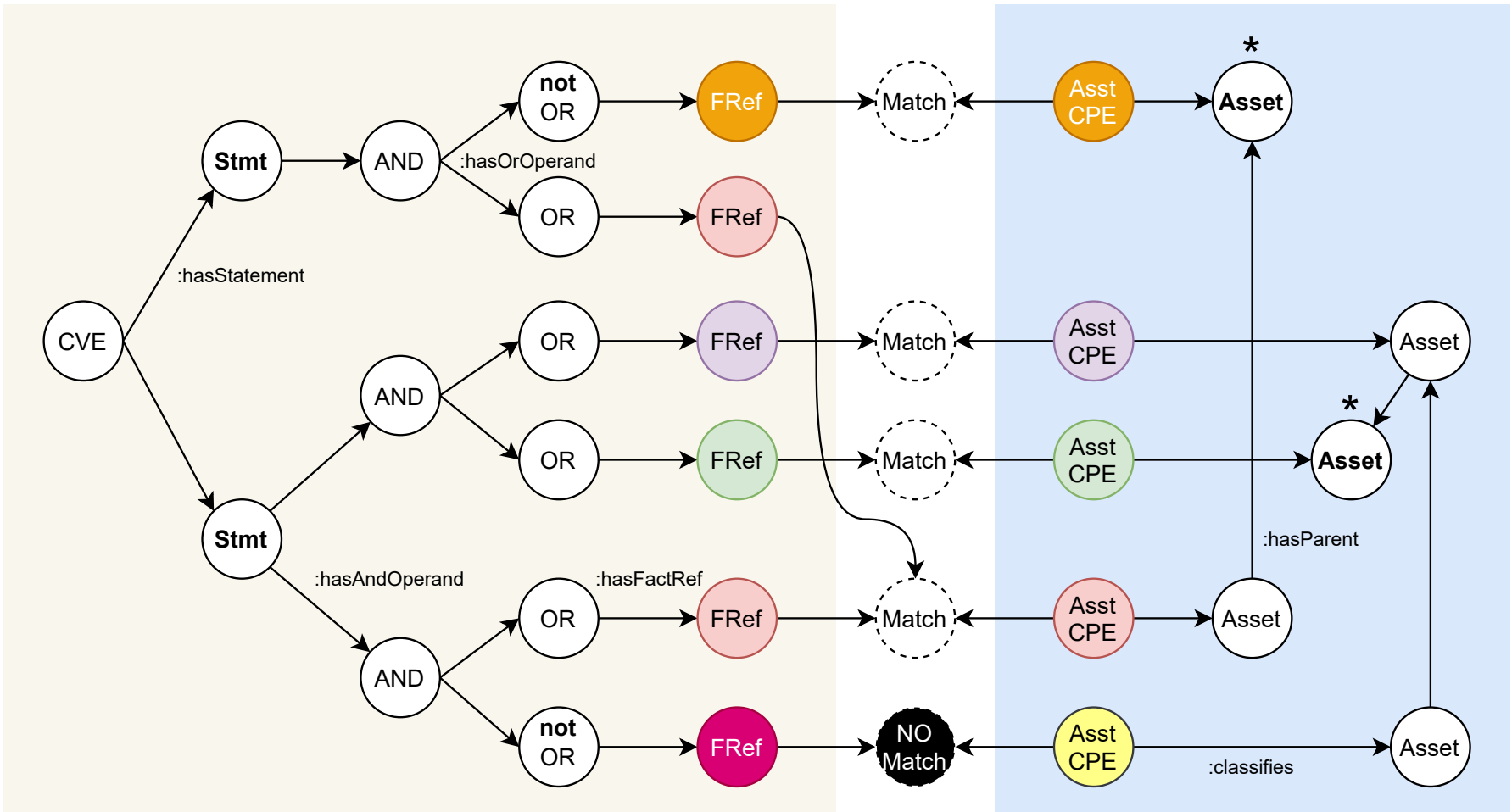
1. For every new CVE vulnerability, `CveVertex` is created and every **applicability statement** it contains, is converted into **CNF**
2. Adjacent `StatementVertices`, `AndVertices`, `OrVertices`, and `FactRefVertices` are recursively inserted.
3. Each `FactRefVertex` is exploded into **11 AV pairs**, i.e. potential future `SourceAVSpecVertices`
4. If a corresponding `SourceAVSpecVertex` **exists**, it is merely **connected** to the `FactRefVertex`, or **created otherwise**
5. If **created**, the new `SourceAVSpecVertex` **is compared** with all the `TargetAVSpecVertices` of the same type (e.g. `version`), and new `RelationEdges` are stored

Insertion Procedure

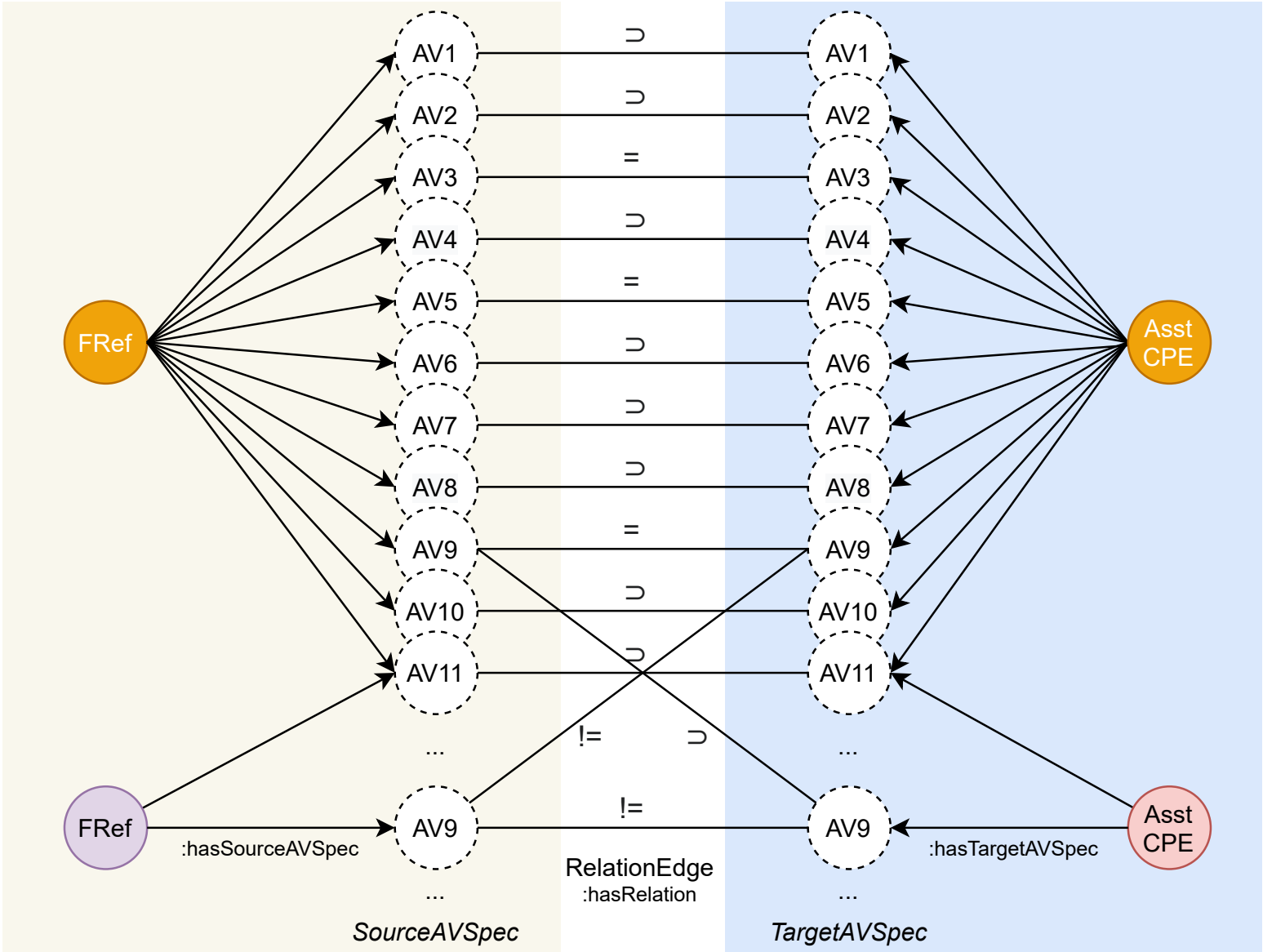
Insert Asset Record

1. `AssetVertex` is **created** for the new asset component and its (optional) **parent is set**
2. Each `AssetCPEVertex` is exploded into **11 AV pairs**, i.e. potential future `TargetAVSpecVertices`
3. If a corresponding `TargetAVSpecVertex` **exists**, it is merely **connected** to the `AssetCPEVertex`, or **created otherwise**
4. If **created**, the new `TargetAVSpecVertex` **is compared** with all the `SourceAVSpecVertices` of the same type (e.g. `edition`), and new `RelationEdges` are stored

Populated Graph Example



Populated Graph Example Detail



Graph Search Query

Starting the Traversal (1)

- **Start** in selected `CveVertices` and continue through the outbound edges to `StatementVertices`, then `AndVertices`, and finally to the `OrVertices`

Branching of the Traversal (2)

- For **positive** literals, find such `(FactRefVertex, AssetCPEVertex)` **pairs that match**
- For **negative** literals, find such `(FactRefVertex, AssetCPEVertex)` **pairs that don't match**
- Uses helper traversals `T1` and `T2`

Finding the Asset Roots (3)

- Find **root** `AssetVertices` for encountered `AssetCPEVertices`
- Parent-less `AssetVertices` represent individual **asset configurations**

Graph Search Query

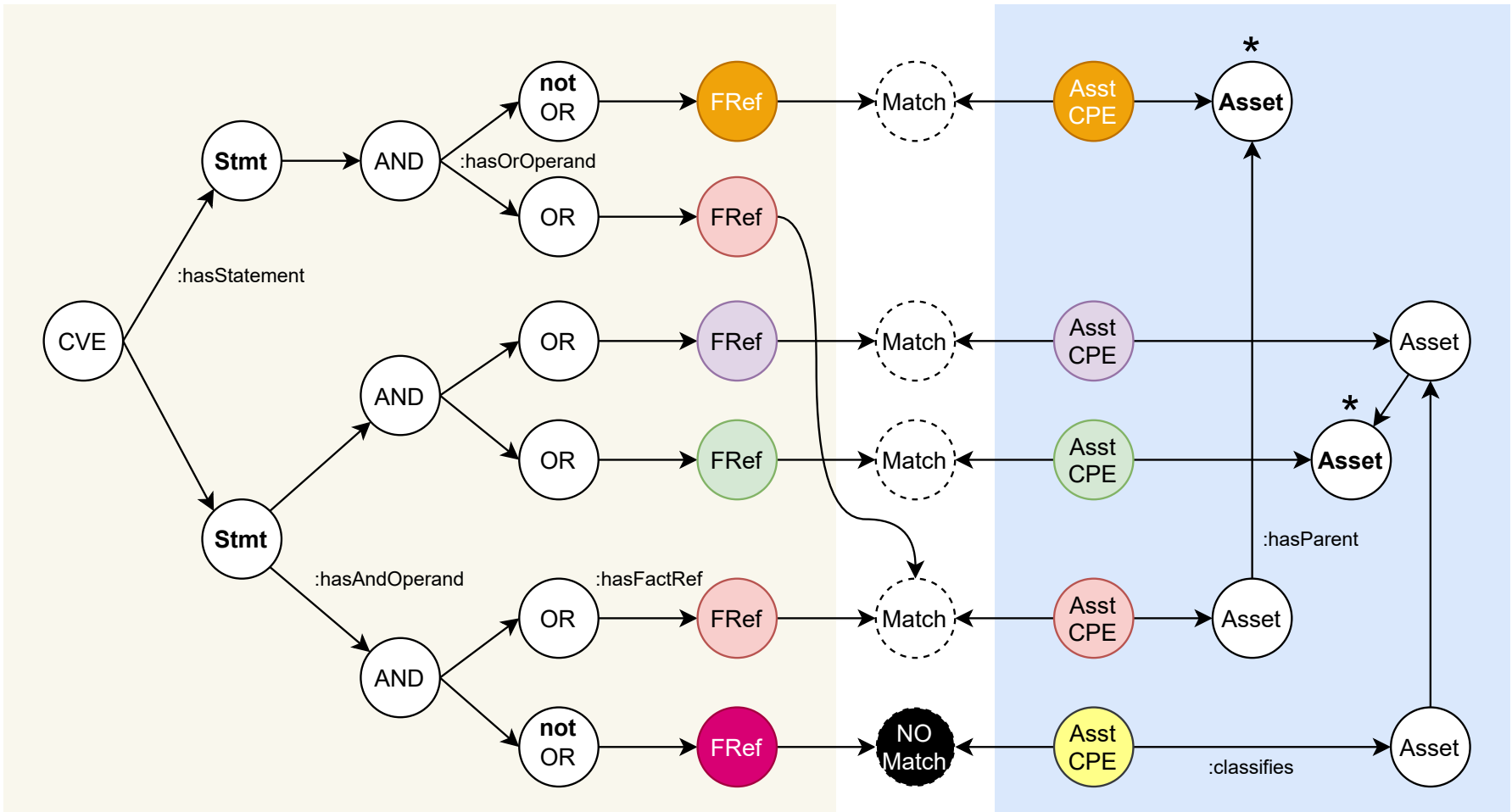
Grouping of the Results (4)

- Graph **traversal stops**
- In the **`group().by()`** step, the traversal history is keyed by the n-tuple of (`CveVertex, StatementVertex, AssetVertex`)
- Each key represents a **candidate match** for a particular `StatementVertex` and a particular `AssetVertex` root

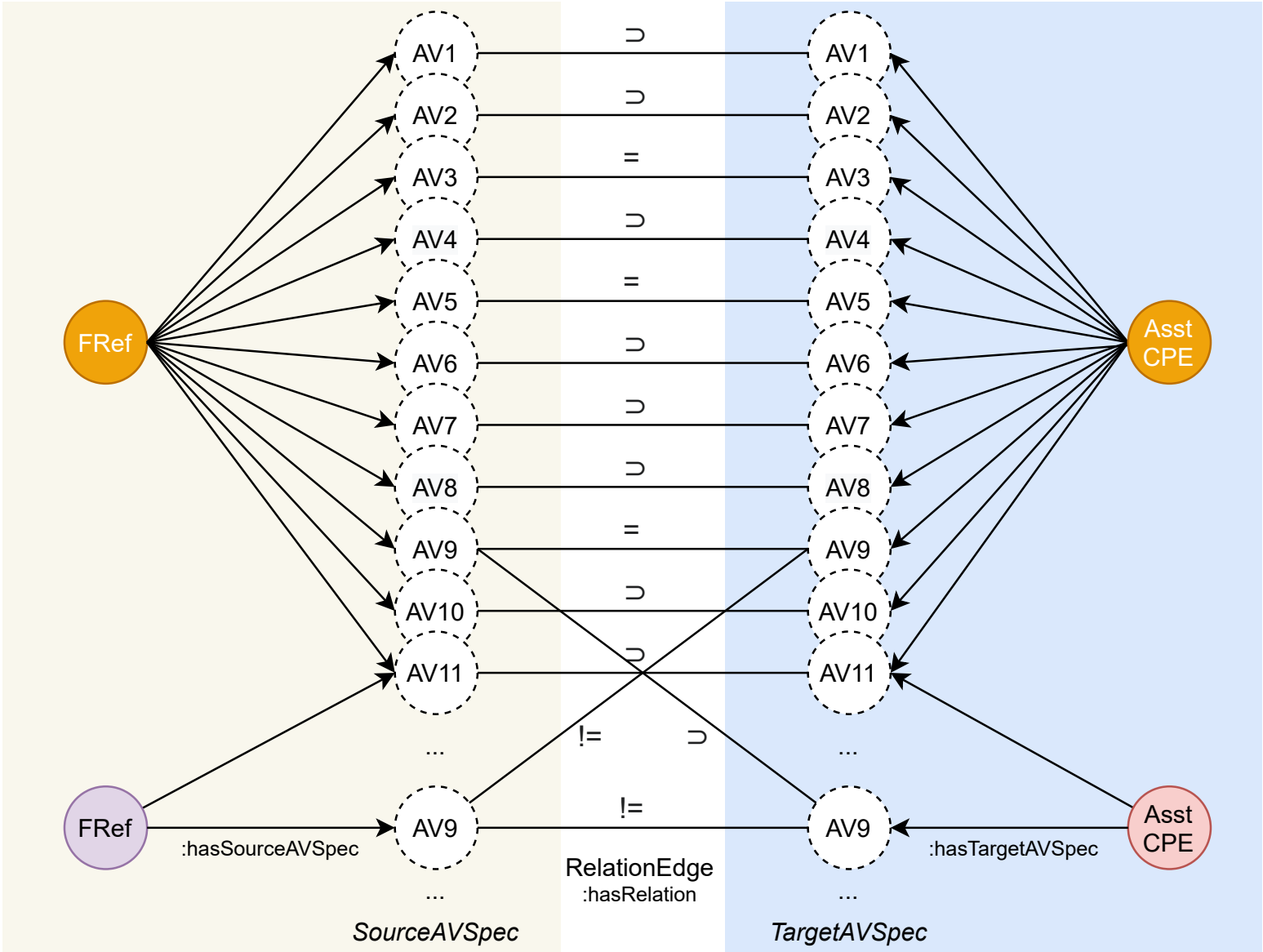
Boolean Evaluation (5)

- Helper traversal `C1` represents the **(expected)** number of `AndVertices` (operands) for each `StatementVertex` (conjunction) **that must hold true**
- Helper traversal `C2` represents the **(actual)** number of AndVertices that **evaluate to true**
- Query returns only grouped pairs where the **counts are equal**

Populated Graph Example



Populated Graph Example Detail



Graph Search Query

Gremlin

- Graph **traversal machine** and **language** (DSL)
- Independent from a particular graph system (database) implementation
- Imperative **traversal queries** and declarative **pattern-match queries** within the same framework

Gremlin Graph Traversal Machine

- Graph G (data), traversal Ψ (instructions), set of traversers T (read/write heads).
- Collection of traversers in T move about G according to the instructions specified in Ψ.
- The computation is complete when either:
  1. There no longer exist any traversers in T
  2. All existing traversers no longer reference an instruction in Ψ

Graph Search Query

Gremlin Graph Traversal Language

- Language enabling a human user to easily define Ψ

Examples

- `out("label")`: Move to the outgoing adjacent vertices given the edge labels.
- `in("label")`: Move to the incoming adjacent vertices given the edge labels.
- `bothE("label")`: Move to both the incoming and outgoing incident edges given the edge labels.
- `hasLabel("label")`: Remove the traverser if its element does not have any of the labels.
- `hasId(<predicate P>)`: Remove the traverser if the element id does not satisfy the given predicate.

Graph Search Query Traversal

```
1  QUERY = g.V()
2  .hasLabel("type::CveVertex").hasId(P1).as_("x_Cve") ①
3  .out("hasStatement").as_("x_Statement")
4  .out("hasAndOperand").out("hasOrOperand").as_("x_Or")
5  .choose(has_("negate", False), T1, T2).as_("x_AssetCPE") ②
6  .dedup("x_FactRef", "x_AssetCPE")
7  .out("classifies").optional(Z).hasId(P2).as_("x_RootAsset") ③
8  .group() ④
9      .by(select("x_Cve", "x_Statement", "x_RootAsset"))
10     .by(select("x_Or", "x_FactRef", "x_AssetCPE").fold())
11 .unfold()
12 .project("match", "path", "expectedCount", "actualCount") ⑤
13     .by(select(Column.keys)).by(select(Column.values))
14     .by(C1).by(C2)
15 .where("expectedCount", P.eq("actualCount"))
```

Graph Search Query Traversal Helpers

```
P1 = P.without([]) # or P.eq(<someCveId>)
P2 = P.without([]) # or P.eq(<someAssetUUID>)
RL = [SUPERSET, EQUALS] # desired AV pair match

M[] = CPE_ATTR_NAMES.forEachElement(_X -> # 11 WFN attributes
__.as_("x_FactRef").out("hasSourceAVSpec").has_("attrName", _X)
.bothE("hasRelation").has_("relation", P.within(RL))
.outV().in_("hasTargetAVSpec").as_("x_Same_AssetCPE")).toArr()
T1 = __.out("hasFactRef") # negate = False
.match([traverse(p) for p in M]).select("x_Same_AssetCPE")

T2 = __.out("hasFactRef").as_("x_FactRef") # negate = True
.out("hasSourceAVSpec").bothE("hasRelation")
.has_("relation", P.without(RL)).outV().in_("hasTargetAVSpec")

Z = __.repeat(out("hasParent"))
.until(outE("hasParent").count().is_(0))

C1 = __.select(Column.keys).select("x_Statement")
.out("hasAndOperand").count()
C2 = __.select(Column.values).unfold().select("x_Or")
.dedup().in_("hasOrOperand").dedup().count()
```

Experimental Implementation

- **Open source implementation** in Java[1]
- Object-Graph Mapping used to implement the graph model
- Search query described in **Gremlin** (for Java)
- **CPE 2.3 Reference Implementation**[2] provides a Java API for creating, using, and matching CPE Names
- Support for AV pair versions defined as **version ranges**

[1] https://doi.org/10.5281/zenodo.4569393
[2] https://pages.nist.gov/cpe-reference-implementation/

Summary

Problem

- Automated vulnerability management via CPE matching

Contribution

- Elimination of **exhaustive search**
- **Graph model** for storing CVEs and their CPE applicability statements and assets with CPE definitions
- **Insertion procedure** that performs AV pairwise comparison
- **Graph search query** that finds all the vulnerable (`CveVertex`, `StatementVertex`, `AssetVertex`) tuples in a **single pass**
- **Experimental implementation** in Java and Gremlin

Future Work

- **Performance evaluation** of the experimental implementation
- Approaches for **asset discovery** and their CPE classification

Acknowledgements

- This research was supported by the Security Research Programme of the Czech Republic 2015–2022 (BV III/1-VS) granted by the Ministry of the Interior of the Czech Republic under No. VI20202022164 Advanced Security Orchestration and Intelligent Threat Management

MASARYK
UNIVERSITY