تمثيل شيفرة المصدر بصيغة RDF واستخدامها في الاستدلال المنطقي عبر Reasoner دلالي لتحديد المتحولات المشبوهة

طالبة الماجستير: م.نورة تركاوي

قسم هندسة النظم والبرمجيات – كلية الهندسة المعلوماتية – جامعة البعث

إشراف: أ.د.محسن حسين

الملخص

انتشرت تطبيقات الويب بشكل كبير وأصبحت عصب الحياة، وزاد الاعتماد عليها في كل مجالات الحياة في ظل الانفتاح الذي يشهده العالم في الوقت الحالي، ومع كل المزايا التي تقدمها شبكة الانترنت تظهر بعض التحديات التي تدفع المطورين إلى إيجاد الحلول لها، ومن أهم هذه التحديات تنوع التطبيقات من حيث اللغة المكتوبة بها إضافة إلى جعلها آمنة وموثوقة، وقد ظهرت العديد من التقنيات التي تبحث في هذه المجالات.

وفي بحثنا هذا انطلقنا من الويب الدلالي الذي ظهر برؤيته التي تعتمد على فهم مدلولات ومعاني البيانات واعتماده على تمثيل البيانات من مختلف المصادر وجعلها بصيغة مفهومة من قبل البشر والآلة على حد السواء، مما جعل من المجدي العمل على إيجاد آلية لتحويل شيفرة المصدر لتطبيقات الويب إلى هذه الصيغة لتخطي حاجز اختلاف لغات البرمجة، ولتحقيق ذلك وانطلاقاً من شيفرة المصدر وملف تعريف اللغة الخاص بلغة شيفرة المصدر سنقوم باستخدام الأداة ANTLR لتوليد شجرة التحليل parse tree والتي تسمى(CST (Concrete Syntax Tree، ومن ثم تحويل هذه

الشجرة إلى صيغة وسيطة XML، وعبر مجموعة من الخوارزميات سنصل إلى ملف شيفرة المصدر مكتوب بصيغة RDF.

واستفدنا في بحثنا من هذه الصيغة في معالجة أحد التحديات الأمنية الكامن في ضعف التحقق من بيانات الدخل اعتماداً على تقنيات الاختبار المتمثلة بالتحليل الستاتيكي للكود عن طريق تمثيل مجموعة من نماذج الكود التي تشكل نقاط دخول محتملة قد تهدد النص البرمجي وفق صيغة RDF وصياغة القواعد التي تصفها هذه النماذج، ومن ثم الاعتماد على تقنيات الويب الدلالي في الاستدلال(Reasoning) لتحديد المتحولات المشبوهة(Tainted) والتي قد تشكل خطراً على الكود.

كلمات مفتاحية: الويب الدلالي، الاستدلال المنطقي، لغة توصيف المصادر، شيفرة المصدر، شجرة التحليل، التحليل الستاتيكي.

Representing the source code in RDF format and using it in Inference via Reasoning to determine Taint variables

Abstract

Web applications have spread widely and have become the backbone of life, and reliance on them has increased in all areas of life in light of the openness that the world is witnessing at the present time, and with all the advantages offered by the Internet, some challenges appear that push developers to find solutions to them, and among the most important of these challenges is the diversity of applications In terms of the language it is written in as well as making it safe and reliable, many technologies have emerged that research these areas.

In our research, we set out from the Semantic Web, which emerged with its vision that depends on understanding the implications and meanings of data and its dependence on representing data from various sources and making it in a form understandable by both humans and machines, which made it feasible to work on finding a mechanism to convert the source code of web applications into these The syntax to overcome the barrier of different programming languages, and to achieve this, based on the source code and the language definition file of the source code language, we will use the ANTLR tool to generate a parse tree called CST (Concrete Syntax Tree), and then convert this tree into an XML intermediate form, and through a set of Algorithms We will access a source code file written in RDF format.

we benefited from this formula in addressing one of the security challenges inherent in the weakness of verification of income data based on testing techniques represented by static analysis of the code by representing a group of code samples that constitute potential entry points that may threaten the script according to the RDF format and formulating the rules that these describe Models, and then rely on semantic web techniques in reasoning to identify suspicious variables that may pose a threat to the code.

Keywords: Semantic Web, Reasoning, RDF, Source code, CST, Static Analysis, Tainted Variables.

1.مقدمة

يتسابق مطوروا الويب والشركات البرمجية في تقديم لغات برمجية جديدة وتطوير تطبيقات مختلفة مما يجعل شبكة الويب شديدة التنوع من حيث اللغات الموجودة ضمنها، وهذا التنوع يخلق معه تحديات عديدة خاصة عندما يتعلق الأمر بضبط أمن التطبيقات وحمايتها من هجمات المخترقين، وبالتالي ومع تعدد التطبيقات وتعدد البيئات واللغات المكتوبة بها، فإنه من الصعب إيجاد حل قابل للتطبيق بشكل فعال وعلى جميع التطبيقات، وفي إطار السعي لإيجاد الحل ووفقاً لمشروع SHIELDS[1] الذي يهدف إلى ردم الهوة بين الخبراء الأمنيين ومطوري التطبيقات عبر تقليل فرص وجود الثغرة أثناء كتابة التطبيق [14]، فإن إحدى الخيارات لتحقيق ذلك يكمن في عملية فحص الكود (Inspection) أو الاختبار القائمة على مراجعة الكود، والتي يندرج التحليل الستاتيكي للكود ضمنها، وهو يقوم على اختبار منطق الكود وسلوكه من أجل كل مسار ممكن من مسارات التطبيق مثل المترجم(compiler)[2].

ولتطبيق التحليل الستاتيكي في ظل تعدد اللغات المستخدمة في كتابة تطبيقات الويب، فإن إيجاد صيغة معينة لتمثيل هذه التطبيقات يعتبر خطوة مساعدة، وبوجود الويب الدلالي – الذي يتيح عمليات الاستدلال المنطقي – اتجهنا لتكون هذه الصيغة هي RDF، لذلك سنقوم في هذا البحث بتقديم الخوارزميات اللازمة لعملية تحويل النص البرمجي المكتوب باللغات المختلفة إلى هذه الصيغة تمهيداً لاستخدامها في التحليل الستاتيكي موضحين كيفية الاستفادة منها في هذا المجال.

1. https://torsec.github.io/shield-h2020/

2.هدف البحث

يتركز هدف البحث في تقديم وسيلة لمواجهة أحد التحديات التي أوجدها تنوع اللغات البرمجية المستخدمة في كتابة تطبيقات الويب مثل جافا(Java) و بي اتش بي(PHP)، وذلك عبر صياغة شيفرة المصدر المكتوبة بلغة ما عبر صيغة توصيف المصادر RDF الخاصة بالويب الدلالي، مما يوفر استقلالاً عن اللغة وبدرجة معينة من الحبيبية مما يسمح باستخدامها في مواجهة نوع آخر من التحديات الكامن في التحقق من متحولات الدخل للتطبيق وتحديد مدى وجود متحولات مشبوهة قد تتدفق عبر التطبيق مسببة حدوث خلل أمني، ويتم ذلك باستخدام نماذج الكود(Code Patterns) التي تشكل نقاط الدخول للكود بعد القيام بتمثيلها وفق الصيغة RDF، ومن ثم صياغة القواعد المقابلة لهذه النماذج بلغة جينا(Jena)، والقيام بالتحليل الستاتيكي للكود عبر الاستدلال المنطقي (Inference) الذي يوفره الويب الدلالي.

3.أدوات البحث:

في سبيل تحقيق هدف البحث لا بد من الاستعانة بالأدوات واللغات المساعدة التالية:

- الأداة Antlr[1] وهي أداة تقوم بتوليد محلل(parser) ليقوم بقراءة ومعالجة وترجمة ملفات النصوص البنيوية أو الثنائية، ويستخدم عادة في بناء لغات البرمجة وأدوات منصات البرمجة[13].
- ملفات قواعد اللغات المختلفة، تم الحصول عليها من الموقع الخاص بهذه الأداة [13] ويمكن إيجادها أيضاً على موقع GitHub[2].

1. https://www.antlr.org/
2. https://github.com/antlr/grammars-v4/tree/master/c#

- XQuery[1] هي لغة استعلام وظيفية يمكن استخدامها للبحث واستخراج العناصر والخصائص من ملفات XML، وقد أصبحت من توصيات W3C منذ عام 2007[15].
- Jena[2] هي أداة ويب دلالي رائدة يستخدمها مبرمجو الجافا [16]، وهي إطار عمل مفتوح المصدر تقدم مجموعة من المكتبات المرنة الخاصة بالاستدلال (inference) باستخدام تقنيات الويب الدلالي.

4.التحليل الستاتيكي

التحليل الستاتيكي: هو تقنية تقوم بفحص الكود البرمجي لتقييمه من الناحية الأمنية [1]. وإن المبدأ الأساسي في تقنية التحليل الستاتيكي هو اختبار وحدات الكود وفحصها للتأكد من تحقيقها للسياسة الأمنية التي تضمن خلو الكود من الثغرات، ويوجد العديد من الحلول والأدوات التي تعتمد هذا المبدأ سنستعرض بعضاً منها لبيان الدافع وراء بحثنا:

يعتبر **Taint Mode** أو ما يقال له الوضع المشتبه أحد الحلول التي تطبق التحليل الستاتيكي، وهو نوع من الفحص توفره كل من اللغات بيرل(Perl) وروبي(Ruby)، وقد تم تصميمه لزيادة الأمان عبر منع المخترقين من تنفيذ الأوامر على جهاز الكمبيوتر المضيف، ويقوم بتسليط الضوء على المخاطر الأمنية التي تنجم عن هجوم الحقن SQLIA (SQL Injection Attack)، وكذلك هجوم امتلاء المخزن، وإن مصطلح Taint يعبر عن أن أي متحول يمكن تعديله من قبل مستخدم خارجي للنظام مثل حقل الإدخال يشكل خطراً محتملاً على التطبيق، فإذا تم استخدام هذا المتحول ضمن تعبير ما أو تم استخدامه لإسناد قيمة لمتحول آخر، يصبح المتحول الآخر أيضاً مشتبهاً به (Taint)، وإذا تم استخدام إحدى هذه المتحولات الموسومة ب Taint في تنفيذ أمر هام مثل تعليمات

1. https://www.w3schools.com/xml/xquery_intro.asp
2. https://jena.apache.org/

Sql على قاعدة البيانات أو على كمبيوتر (نظام تشغيل)، يقوم فاحص ال Taint بإصدار تحذير بأن البرنامج يستخدم متحول محتمل الخطورة [6].

ويعتبر تحليل taint طريقة فعالة لتقليل المخاطر من خلال المساعدة في اكتشاف ومنع نقاط النصوص البرمجية القابلة للاستغلال، نظراً لأن المتحولات المشتبه بها يمكن أن تتدفق عبر البرنامج بطرق غير متوقعة، فمن المهم فهم هذه المسارات بالكامل، وعبر تحليل الـ taint يمكن تصور تدفق البيانات المشبوهة مما يساعد المطورين على فهم مخاطر الكود الخاص بهم ويساعدهم في تحديد أفضل السبل لتغيير الكود لإغلاق الثغرة الأمنية، ويوفر هذا النمط استمرارية في الأمن للتطبيق حيث أن النصوص البرمجية تكون عرضة للتطوير بشكل مستمر مما يعني تغيراً في الكود وإضافة ميزات جديدة، وعلى الرغم من أن الكود الأصلي قد يكون آمناً تماماً، فقد يقوم شخص ما بتغيير كل ذلك عن طريق الخطأ، ففي هذه الحالة يمكن اعتبار نمط ال taint بمثابة تدقيقٍ أمنيٍ مستمرٍ مجاني[6].

و بالرغم من أن هذا النمط فعال وموثوق في إيجاد الثغرات بشكل أوتوماتيكي من أجل بعض أنواع التهديدات مثل SQLI , Buffer Overflow ، إلا أنه يعاني من بعض المحدودية:

- ارتفاع نسبة false Positives
- من الصعوبة تحديد فيما إذا كانت الحالة الأمنية المكتشفة تشكل فعلاً تهديداً، حيث أنه قد يتم وسم المتحول بأنه Taint وهو لا يشكل خطراً حقيقياً.
- ليست مستقلة عن اللغة، حيث لا يمكن تطبيق Taintالخاصة بلغة Perl على تطبيقات مكتوبة بلغة Ruby.

ويشكل توسيع اللغة Language Extension حلاً آخر لتطبيق التحليل الستاتيكي، وهنا يتم توسعة اللغة عبر إضافة annotations مهمتها دعم السياسة الأمنية التي تحافظ على سلامة تدفق المعطيات عبر التطبيق، حيث تقوم هذه الإضافات بالفحص الستاتيكي لكل احتمالات سير وتدفق البيانات، وتعتبر JFlow مثالاً على ذلك فهي لغة جديدة تقوم بتوسعة لغة الجافا لحماية سرية وسلامة البيانات الحساسة في التطبيق، وذلك عبر استخدام توكيدات(assertions) هدفها منع تسرب المعلومات المهمة بطريقة حسابية، حيث يتم إحاطة المتحولات أثناء التصريح عنها بالشروحات(annotations) و التي هي عبارة عن لافتات اسمية(labels) خاصة بها، وتسمح للفاحص الستاتيكي بفحص البرنامج والتأكد من خلوه من الأخطاء التي تؤدي لتسريب البيانات[7]، وتمتاز هذه الطريقة بأن المترجم الخاص بال JFlow هو عبارة عن فاحص ستاتيكي يقوم بترجمة الكود المكتوب بالـ JFlow إلى الكود المقابل بالجافا، وهذه العملية تتم بشكل ستاتيكي فلا تشكل عبئاً أثناء التشغيل، ولكن لا يمكن تطبيقها على لغات أخرى غير الجافا وهذه تعتبر مشكلة حقيقية تستدعي الحل، وهناك نوع آخر من الحلول يتم فيه تقديم لغة سكريبت جديدة تماماً لضمان سرية وسلامة البيانات، هذه الطريقة أيضاً تعاني من محدودية كونها خاصة بلغة معينة ولايمكن تطبيقها على كل التطبيقات كما تُعتبر غير موثوقة على المستوى الأمني.

ولقد تبين لنا أثناء البحث أن الحلول الموجودة التي تطبق تقنية التحليل الستاتيكي والتي يتم استخدامها لضمان سلامة الكود وخلوه من الثغرات تعاني من بعض المحدودية ولايمكن اعتبارها حلولاً شاملة وفعالة، حيث أن بعضها يقدم لغة جديدة مما يحول دون تطبيقها على الأنظمة الموجودة ويتطلب إعادة كتابة الكود بما يتوافق مع اللغة الجديدة المقدمة، ومن ناحية أخرى فإن بعض الحلول تكون مصممة لتطبيقات بلغة محددة ولايمكن تطبيقها على تطبيقات مكتوبة بلغة أخرى، وبما أن تطبيقات الويب الموجودة حالياً شديدة التنوع من حيث اللغات المستخدمة في برمجتها فكان لابد من إيجاد صيغة

مناسبة لتمثيل شيفرة التطبيق مما يتيح لنا فيما بعد إمكانية استخدام هذه الصيغة للقيام بالتحليل الستاتيكي.

وبالنتيجة فقد اتجهت بعض التطبيقات الحالية لاعتماد تقنيات أكثر ذكاء كأن تكون الصفحات قابلة للفهم من قبل الآلة، فظهر الويب الدلالي (Semantic Web SW) والذي يعتبر ثورة جديدة في عالم الويب ومتصفحات الإنترنت[3]، ويهدف الويب الدلالي حسب رؤية منشئه ومنشئ الويب Tim Berners–Lee إلى تحويل الكم الهائل من البيانات ومصادر المعلومات المتاحة على الشبكة العالمية من مجرد وحدات مكونة من صفر وواحد (نظام البتات) إلى بيانات مفهومة من قبل برامج الحواسيب التي تنشأ خصيصاً لهذا الهدف، وهو محاولة لتطوير لغة للتعبير عن المعلومات والبيانات في شكل قابل للقراءة والمعالجة آلياً، والفهم ليس فقط من قبل البشر بل من قبل الآلات أيضاً (الحواسيب)، ويقوم على مبدأ الوجوديات والتي تعني تمثيل المعرفة على أنها مجموعة من المفاهيم والعلاقات بينها في مجال معين [3]، وتعتبر لغة XML و XHTML واللغات المتطورة الأخرى هي نقاط بدء جيدة لإنجاز هذا العمل.

وتعتبر تقنيات الويب الدلالي واعدة في مجال تطبيق التحليل الستاتيكي للكود لكشف الثغرات[3]، وهناك العديد من الأبحاث التي اتجهت لتحقيق هذه الفكرة نورد فيما يلي نموذجين عنها:

4-1 التحليل الستاتيكي المبني على الموديل للثغرات الأمنية(Ontology Model-Based Static Analysis of Security Vulnerabilities)

تقدم هذه الوثيقة[11] خوارزمية للتحليل الستاتيكي تعتمد على موديل أنطولوجي محسن عبر تقنيات التقطيع (program slicing) لكشف الثغرات، وتتم آلية العمل عبر ثلاثة مراحل:

1. بناء مكتبة المعارف: وهنا يتم بناء النماذج المطلوبة لتوصيفات الجافا (java specifications) عبر الowl ، وللثغرات باستخدام لغة القواعد SWRL.
2. نمذجة كود المصدر: ويتم في هذه المرحلة العمليات التالية:
   - تحليل الكود وبناء شجرة AST.
   - بناء مخططات اعتمادية PDG للطرائق الموجودة في الكود وSDG للنظام الكلي.
   - التقطيع عبر البيان وفقاً لمعايير التقطيع التي تم استخراجها من قواعد SWRL، حيث يتم استبعاد عناصر البرنامج التي لا صلة لها بهذه المعايير ولا تؤخذ بعين الاعتبار في المرحلة التالية. كمثال: لا يجوز استدعاء طريقة start للمسلك من داخل الباني.
   - عمل إسقاط للعقد المقطوعة إلى مفردات ال OWL المقابلة لها.
3. الاستدلال: يتم عمل تحليل للمرحلة السابقة للكشف عن وجود الثغرات أو عدمه وإصدار التقارير المطلوبة.

وتمتاز هذه الطريقة بأنها تدعم التوسعة للغة عبر نمذجة التوصيفات حيث يمكن توسيع النموذج بقدر إضافة التوصيفات بشكل سهل، إلا أنها تعاني من كثرة المفردات والتي تنتج عن نمذجة كل التوصيفات وربط بين كل عناصر الجافا إلى مفردات الانطولوجي خلال المرحلة الثانية.

2-3-4 التزويد بنموذج تحليل أمني لشيفرة المصدر باستخدام تقنيات الويب الدلالي(Providing a Source Code Security Analysis Model Using Semantic Web Techniques).

لتغطية مشكلة تعدد اللغات المستخدمة لكتابة شيفرة المصدر لتطبيقات الويب تم استخدام خوارزمية تعتمد على كشف نمط التصميم للكود (design pattern) وبناء نموذج انطولوجي يمثل عناصر الكود والعلاقات بينها[12]. وتتألف آلية العمل من مرحلتين:

1. التحليل(parsing): يتم أخذ الكود كدخل، وبناء شجرة المفردات المجردة AST له، ثم اكتشاف مكونات الكود والعلاقات بينهم عبر التنقل بين عقد الشجرة وإنشاء ملفات الN3 المؤلفة من ثلاثيات RDF، ثم يتم تخزينها في المستودع الخاص بالثلاثيات.
2. الاستدلال: يتم وضع قواعد لأنماط الثغرات التي نريد الكشف عنها، ثم يتم تشغيل استعلام SPRQL وفق هذه القواعد والثلاثيات المخزنة في المستودع من المرحلة السابقة ويتم إيجاد نقاط الخلل وإصدار التقارير المطلوبة.

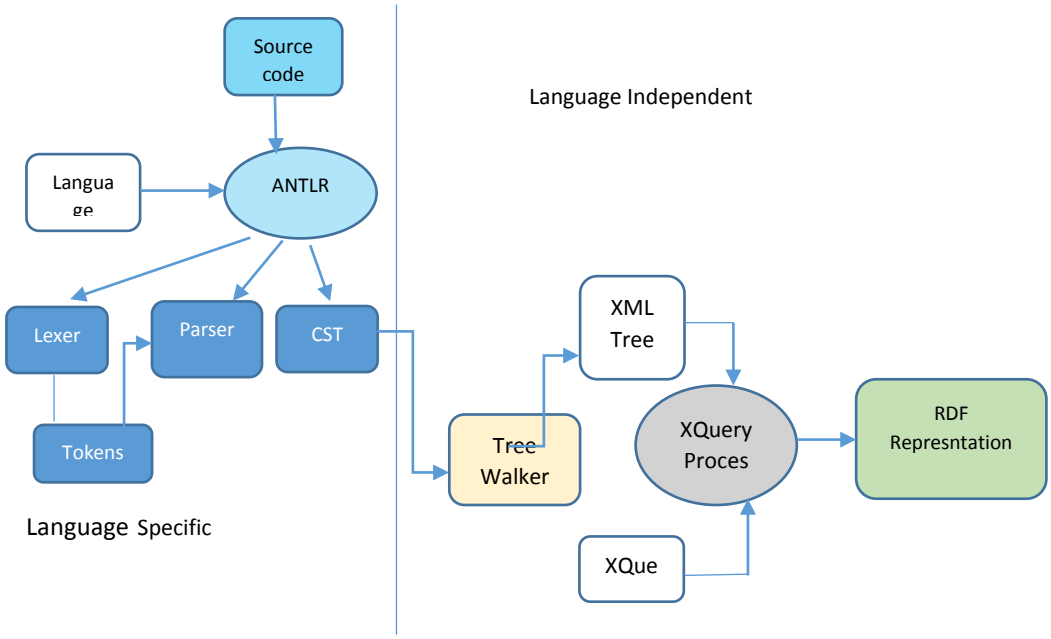
وتمتاز هذه الطريقة بعدم وجود اعتمادية بين مرحلة تحليل الكود ومرحلة اكتشاف الأخطاء، حيث يمكن تطوير كل واحدة بشكل مستقل عن الأخرى، ولكن جميع الأدوات المستخدمة تعتمد على الجافا (Jtest tool – Find Bugs – J2RDF) في كلا المرحلتين.

لذلك وبعد إثبات قدرة تقنيات الويب الدلالي سنتجه في بحثنا للعمل على إيجاد طريقة للاستفادة من هذه التقنيات وقدراتها المتقدمة والتي تتجاوز حدود اللغات البرمجية، وسيتم ذلك عبر تحويل النص البرمجي المكتوب به تطبيق الويب إلى الصيغة المستخدمة ضمن الويب الدلالي وهي RDF، و التي هي عبارة عن جمل كل جملة منها تشكل ثلاثية من الشكل S–P–O المبينة بالشكل[11]، وعملية التحويل ليست بهذه البساطة فهي مؤلفة من مرحلتين وكل مرحلة مؤلفة من عدة خطوات وسيتم استخدام مجموعتين من الخوارزميات الأولى وظيفتها تحويل النص البرمجي إلى الصيغة الوسيطة XML والثانية تحول ملف ال XML إلى ملف RDF ليتشكل لدينا ملفات RDF جاهزة للعمل، ومن ثم يمكن وضع نماذج الكود التي تشكل ثغرات ممكنة والتي تكون مكتوبة بلغات برمجية مختلفة مما يستدعي تمثيلها وفق الصيغة المختارة باستخدام ثلاثيات ال RDF، وانطلاقاً من هذه الثلاثيات يمكن وضع قواعد مصاغة باستخدام لغة Jena لاستخدامها في عمليات الاستدلال المنطقي(Reasoning) وبالتالي إجراء عملية التحليل الستاتيكي

للنص البرمجي بعد تحويله أيضاً إلى الصيغة RDF، ونتيجة عملية الاستدلال تبين مدى تعرض التطبيق للثغرة المدروسة.

5.أداة تحويل النص البرمجي إلى RDF

كما رأينا الحلول الموجودة غير قابلة للتطبيق على معظم تطبيقات الويب المكتوبة بلغات متنوعة، لذلك نحن في بحثنا نهدف للوصول لاستخراج حقائق المصدر مصاغة بالـ RDF، وبالتالي بشكل مستقل عن اللغة وبدرجة معينة من الحبيبية أي بحجم مناسب حيث أنه احياناً لا يكفي سطر واحد من الكود لتحديد وجود ثغرة أم لا بل يتطلب كتلة برمجية كاملة، وتتألف عملية استخراج الحقائق من شيفرة المصدر من مرحلتين: مرحلة مرتبطة باللغة وأخرى مستقلة عنها الشكل[1].



الشكل (1) مراحل استخراج حقائق كود المصدر

5-1 المرحلة المرتبطة باللغة:

وتنطوي هذه المرحلة على مجموعة من الخطوات تبدأ باستخدام الأداة Antlr حيث سنزودها بملفات قواعد اللغات المستخدمة في كتابة تطبيقات الويب إضافة إلى ملفات شيفرة المصدر، وستكون النتيجة ملفات المحلل(parser) والمعجم(lexer)، ويتم استخدام هذه الملفات لتوليد الشجرة parse tree (CST) والتي ستكون خرج هذه المرحلة الشكل(1).

5-1-1 الأداة ANTLR (Another Tool for Language Recognition)

وهي أداة تقوم بتوليد محلل(parser) ليقوم بقراءة ومعالجة وترجمة ملفات النصوص البنيوية أو الثنائية، ويستخدم عادة في بناء لغات البرمجة وأدوات منصات البرمجة، حيث أنه ينطلق من توصيف اللغة أو ما يسمى بملف قواعد اللغة لينتج محللاً لهذه اللغة، ويمكن استخدام هذا المحلل لبناء Parse tree والتي يشار إليها أيضاً ب CST (Concrete Syntax Tree) وهي عبارة عن بنية معطيات تمثل كيف يتم عمل مطابقة بين ملف القواعد والدخل، ويقوم Antlr بإنشاء ما يسمى ب walkers ليتم التنقل على الشجرة وتنفيذ كود ما[4]. ومن الجدير بالذكر أن Antlr يتم استخدامها في تويتر مثلاً لتحليل الاستعلامات وتصل قدرتها إلى بليوني استعلام في اليوم، كما يتم استخدامها في أنظمة التحليل الخاصة ب Hadoop والـ data warehouse، وفي NetBeans يتم استخدامها لتحليل ملفات C++ وفي أوراكل وغيرها، وسنستخدم في بحثنا الإصدار "Antlr 4.7.1".

بالطبع يوجد أدوات أخرى لإنتاج المحللات مثل META,Coco/R,CUP وغيرها، ولكننا اخترنا Antlr للأسباب التالية:

- ❖ يعتبر اختيار Antlr موفقاً من حيث تتبع الأخطاء: حيث أنه يعتبر من أنواع المحللات LL أي(Left-to-right, Leftmost derivation)والذي يكون فيه الإبلاغ عن الأخطاء أكثر قابلية للقراءة.
- ❖ في حين أن المحللات من النوع LR(Left-to-right, Rightmost derivation in reverse) تقدم تقريراً عن الأخطاء أثناء مواجهة تعارض من نوع Shift-Reduce والتي تعد أكثر أنواع التعارضات شيوعاً في القواعد النحوية، ويحدث عند محاولة عمل تقليص قاعدة لمفردة معينة وبنفس الوقت السماح لقاعدة أخرى بعمل إزاحة على نفس المفردة، وهذا يجعل تتبع الأخطاء عملية صعبة وتحتاج لسجل التعقب.
- ❖ يحتوي ANTLR على قواعد نحوية للمعجم(lexer) والمحلل(parser) معًا، باستخدام الأحرف الكبيرة والصغيرة للتمييز بينهما، وهما مكتوبان في نفس النوع من القواعد، في حين أن محللاً آخر مثل CUP قد فصل قواعد lexer(Jlex) و parser (CUP) عن بعضهما، مما يستدعي تعلم شيئين قبل استخدامه.
- ❖ القواعد النحوية لـ ANTLR أسهل في الكتابة، وأسهل في القراءة من القواعد النحوية لغيره مثل CUP، خاصة بعد إضافة الإجراءات في القواعد، على سبيل المثال: تحتاج CUP إلى الإشارة إلى أسبقية الرموز، بينما تحتوي ANTLR على هذه المعلومات وفقًا لبنية القواعد الخاصة بك.

5-1-2 ملفات قواعد اللغة:

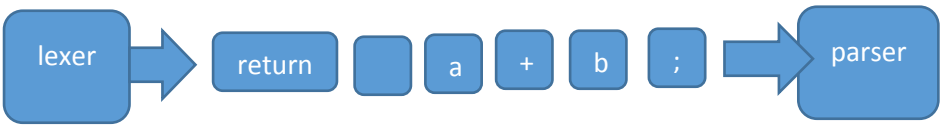
يمكن تحميلها من موقع Antlr الرسمي الذي يوفر وبشكل دائم تحديثاً لملفات قواعد معظم اللغات مثل Pascal ,java , #C, php..... أو يمكن كتابة ملف قواعد

```
// Define a grammar called Hello
grammar Hello;
r  : 'hello' ID ;          // match keyword hello followed by an
identifier
ID : [a-z]+ ;              // match lower-case identifiers
WS : [ \t\r\n]+ -> skip ; // skip spaces, tabs, newlines
```

معرف من قبل المستخدم على النحو التالي:

5-1-3 المحلل(Parser) والمعجم(Lexer)

يتم استخدام Antlr لتوليدهما باستخدام ملف قواعد اللغة وملفات شيفرة المصدر، والهدف من lexers الحصول على حبيبية أدق، أما الـ parser الهدف منه فصل لغة الدخل عن المنطق للتطبيق، وعند عمل ترجمة لهذه الملفات يمكن استخدام ناتج هذه الترجمة لتوليد شجرة التحليل، حيث يقوم الlexer بتصنيف الـ lexems التي تمت قراءتها من ملف شيفرة المصدر، على سبيل المثال فإن "==" و "!=" هي lexems يتم تصنيفها ضمن php بأنها معاملات مساواة(EqualityOperators)، وبالتالي فإن خرج الـ lexerهو عبارة عن سلسلة مفردات(tokens) كما في المثال الموضح بالشكل (2) والموافق للكود: return a+b;

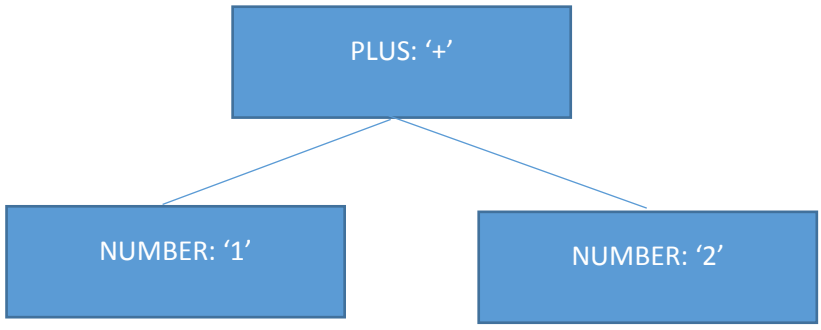


الشكل (2) تسلسل المفردات من ال lexer إلى ال parser للكود return a+b;

ثم يقوم المحلل بقراءة سلسلة المفردات ومطابقتها مع القواعد الموجودة فيه، فعلى سبيل المثال عندما يقرأ المحلل تسلسل المفردات الواردة إليه، وبمقارنتها مع القواعد التالية:

```
* PARSER RULES
*-------------------------------------------------
----------*/
add: NUMBER PLUS NUMBER;
/*-------------------------------------------------
* LEXER RULES
*-------------------------------------------------*/
NUMBER: ('0'..'9')+ ;
```

فإنه يفسر التسلسل 1+2 على أنه عملية (add) جمع لمفردتين من النوع رقم، وبالتالي يتم توليد شجرة التحليل التي يتم اختزالها وصولاً للشجرة المجردة AST الموضحة بالشكل (3) بناء على ذلك.



الشكل(3) شجرة التحليل المجردةASTالمقابلة للتعبير 1+2

وبما أننا في الإصدار الحالي من Antlr لم نحصل على شجرة التحليل المجردة AST فقد اكتفينا بشجرة التحليل التفصيلية CST ويمكن فيما بعد أثناء تطوير خوارزمية إنتاج الشجرة المختصرة، وهو أمر كان الإصدار السابق من Antlr يدعمه ولكن لأسباب تتعلق بالتغيرات التي تطرأ على ملفات القواعد قد تم إلغاؤه في الإصدارات اللاحقة، وبالنسبة للإصدار الذي نستخدمه قد تم مؤخراً تقديم أداة توفر لنا آلية للوصول إلى الشجرة

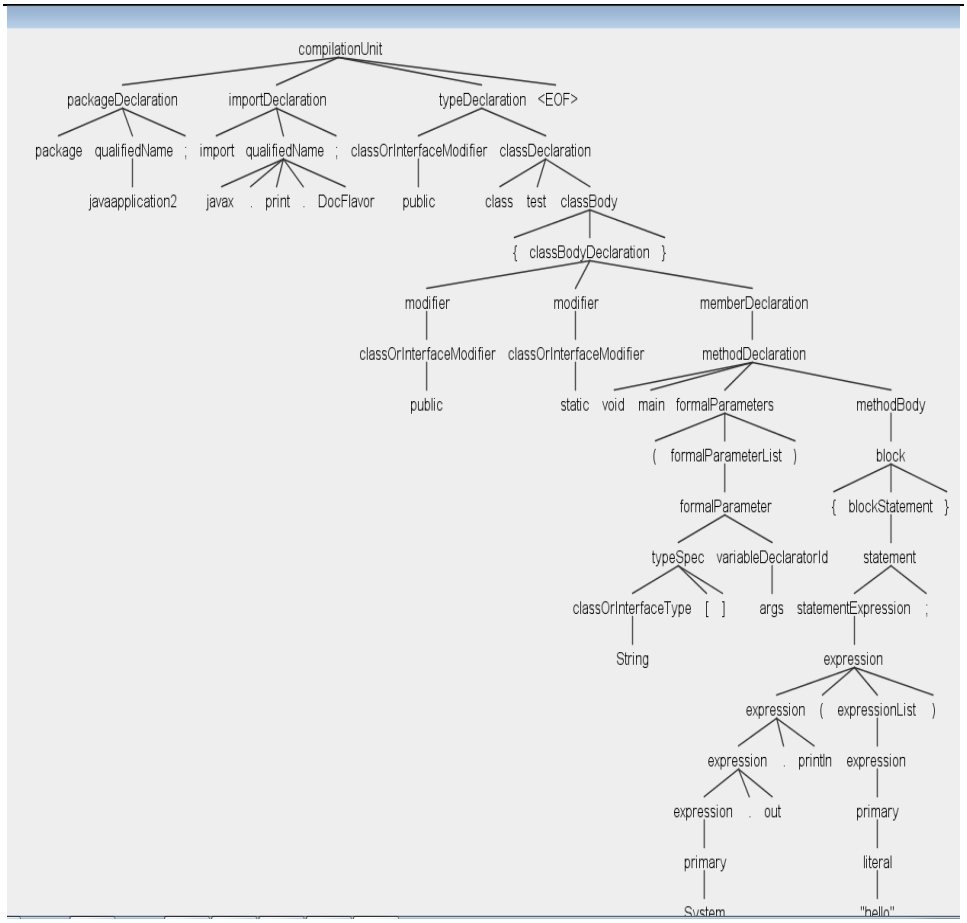
المختصرة[1]، والشكل(4) يبين مثالاً على ملف مصدر مكتوب بلغة جافا مع شجرة ملفات " JavaLexer.java" و " JavaParser.java" التي تم توليدها باستخدام الأداة Antlr مع ملف قواعد جافا

1.https://www.antlr.org/api/JavaTool/org/antlr/v4/tool/ast/package-summary.html

```
package javaapplication2;


import javax.print.DocFlavor;

public class test {
    public static void main(String[] args)
{System.out.println("hello");}
}
```



الشكل(4) ملف الجافا وشجرة التحليل CST المقابلة له

2-5 المرحلة المستقلة عن اللغة:

إن دخل هذه المرحلة هو خرج المرحلة السابقة أي شجرة التحليل، ويتم خلالها التنقل على عقد الشجرة باستخدام خوارزمية تقوم ببناء ملف XML ذو البنية الشجرية كصيغة وسيطة، ثم باستخدام خوارزمية أخرى تزور عقد XML وتستخدم المعالج الخاص بال XQuery لتوجد مقابلاتها في الصيغة الموحدة التي سنطبق عليها التحليل الستاتيكي وفق منطق الويب الدلالي وهي في بحثنا إطار توصيف المصادر RDF الشكل(1)، ولا بد من الإشارة إلى أن RDF يعتبر توصيفاً مناسباً في مجال نشر قواعد البيانات على

الويب ، حيث نحتاج أن نخصص لكل شيء في قاعدة البيانات تعريفاً مناسباً لوظيفته بحيث يستطيع الآخرون التحدث عنه واستخدامه بيسر وفهمه أيضاً، وتتألف RDF من أربعة مركبات رئيسية[8]:

- **Resources** (مصادر): وهي أي شيء يمكن وصفه بال RDF ويمكن أن يكون له URI، يتضمن ذلك صفحات الويب والعناصر الفردية لمستند XML والصور والأشخاص وأي شيء تقريبًا في الكون سواء كان رسميًا على الويب أم لا.
- **Properties** (خصائص): وهي عبارة عن سمة (attribute) مسماة تصف مصدراً، على سبيل المثال: قد يحتوي الكتاب الذي يشكل مصدراً له URI التالي: "12345:urn:isbn" على خاصية المؤلف و / أو عنوان الكتاب، وفي RDF يمكن أن تكون الخصائص مصادراً بحد ذاتها مع خصائصها الخاصة.
- **Value** (القيمة): قيمة الخاصية والتي يمكن أن تكون عبارة عن مجرد سلسلة نصية مثل: "Tim Berners-Lee"، أو قد تكون مصدراً مع مجموعته من الخصائص، وهذا قد يجعل أوصاف الويب الدلالي معقدة للغاية.
- **Statements** (جُمل): مجموعة المصادر مع أسماء الخصائص وقيمها، وهي ممكن أن تعتبر أيضًا مصادراً بحد ذاتها ، مما يعني أنه يمكننا أن ننسب إليها سمات، وتتألف الجملة من الثلاثية:

Subject(S)–**Predicate**(P)–**Object**(O)

- مثلاً الجملة: "The sky has the color blue"
- "Sky": subject "has the color ": predicate "blue ": object

5-2-1 التحويل من شجرة التحليل إلى ملف xml

حتى الآن أصبح لدينا ملفات الـ lexer والـ parser التي استطعنا عن طريقها الحصول على شجرة التحليل، والتي هي عبارة عن عقد كل منها من نمط غرض يمكن الوصول إليه والتعامل معه، وبما أن هدفنا هو الوصول لصيغة RDF الموحدة لنتمكن

من تطبيق التحليل الستاتيكي عليها عبر تقنيات الويب الدلالي، فقد قمنا باستخدام مرحلة وسيطة تتمثل بالوصول لصيغة XML ومنها للصيغة المطلوبة، ويعود سبب اختيارنا لهذه الصيغة إلى إمكانية تمثيل المعلومات الموجودة في الشجرة دون ضياع باستخدام نفس البيئة الشجرية، مستفيدين من بنية ملفات XML الشجرية[10].

تتمتع ملفات XML – والتي تعد من توصيات W3C1 – ببنية شجرية حيث يبدأ المستند بالجذر ويتفرع لينتهي بالأوراق، والشكل(7) عبارة عن مثال يوضح كلاً من بنية الشجرة وبنية ملف XML المقابل لها، حيث أن الجذر في كل منهما هو العقدة 'ASSIGN: 'assign، والأوراق هي العقدة 'VARIABLE: 'a والعقدة 'VARIABLE: 'n والعقدة NUMBER: '1'.

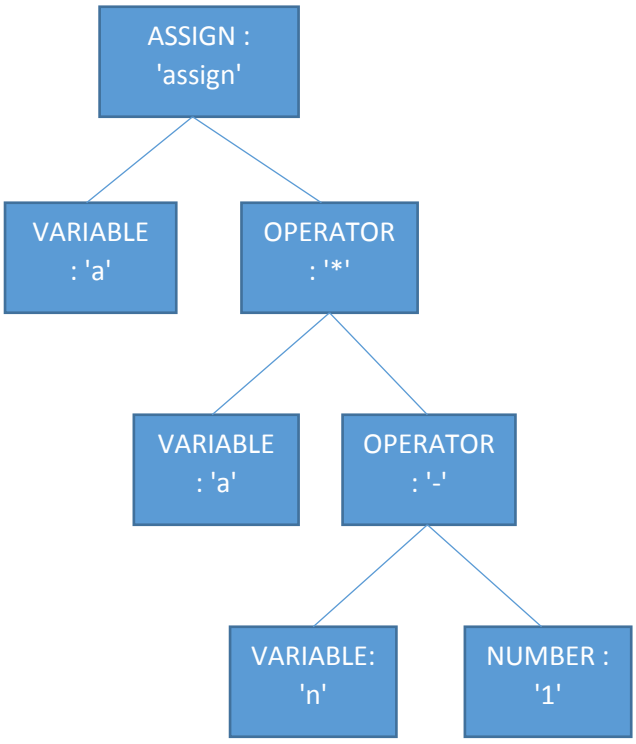
وإن عملية التمثيل لعقد شجرة التحليل تتم عبر زيارة جميع العقد وفق الخوارزمية التي سترد لاحقاً، وتمثيل كل عقدة بالشجرة بعقدة مقابلة لها بال XML، ويكون اسم عقدة XML مقابلاً لاسم المفردة بالشجرة، ومحتوى عقدة XML مقابلاً للنص المكتوب بعقدة الشجرة.

1.https://www.w3.org/standards/xml/core

5-2-2 خوارزمية تحويل شجرة التحليل إلى مستند XML

تم استخدام نموذج من الأعلى للأسفل top-down للتنقل على الشجرة أي زيارة العقد من الجذر وصولاً للأوراق لتحويل شجرة التحليل إلى ملف XML المقابل كما هو موضح في الشكل(5)، حيث تتم زيارة كل عقدة وطباعة النمط والقيمة لأبناء هذه العقدة وفق الصيغة <type>value</type>، ومن أجل كل ابن للعقدة يتم إعادة العملية بشكل

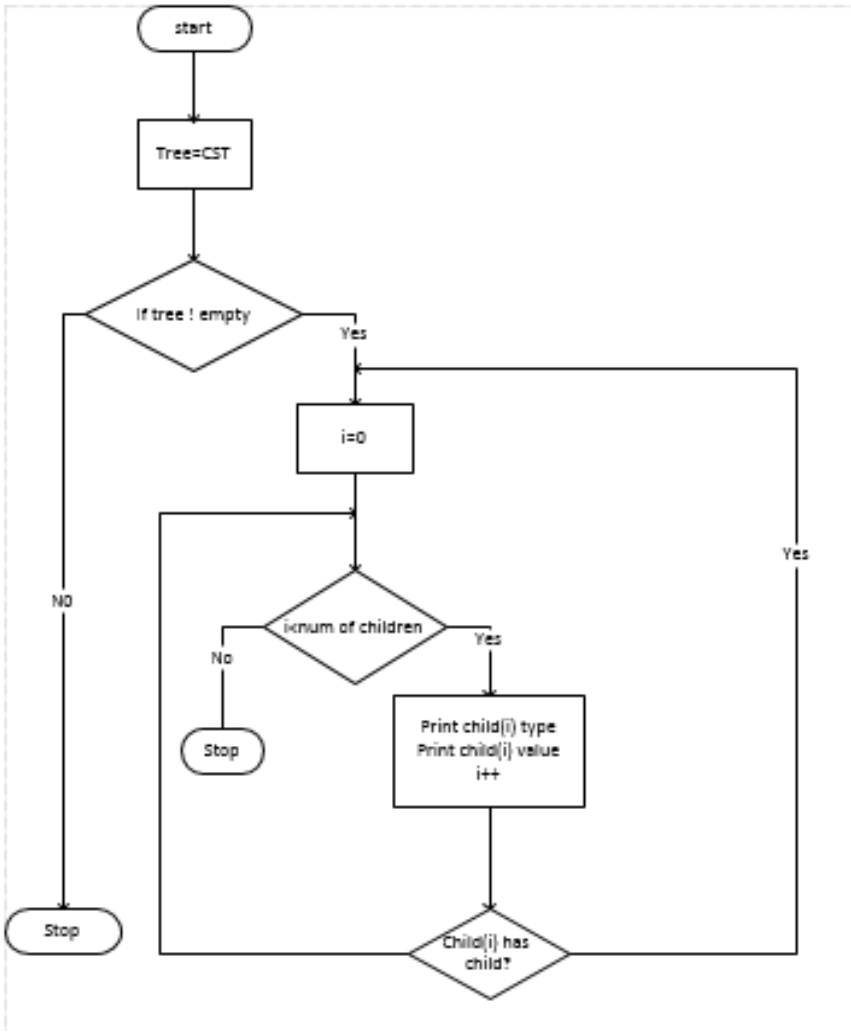
عودي حتى تنتهي كل العقد، ومن أجل البوادئ indentation يتم طباعة فراغ " " وهو ليس ذو معنى وإنما لتحسين الشكل لسهولة القراءة، والشكل (5) يوضح الخوارزمية:




```
<ASSIGN>
    assign
    <VARIABLE>
        a
    </VARIABLE>
    <OPERATOR>
        *
        <VARIABLE>
            a
        </VARIABLE>
        <OPERATOR>
            -
            <VARIABLE>
                n
            </VARIABLE>
            <NUMBER>
                1
            </NUMBER>
        </OPERATOR>
    </OPERATOR>
</ASSIGN>
```

الشكل(5) شجرة التحليل وملف xml المقابل لها

والشكل(6) يبين خوارزمية التحويل من شجرة التحليل إلى XML:



الشكل(6) خوارزمية تحويل CST إلى XML

وعند تطبيق مجموعة التوابع التي تحقق خوارزمية التحويل على شجرة التحليل التي تمثل ملف الجافا الوارد بالشكل(4) نحصل على ملف XML نورد فيما يلي جزءاً منه:

```
<?xml version="1.0" encoding="UTF-8"?><CompilationUnit>
<PackageDeclaration>
package
<QualifiedName>
javaapplication2
</QualifiedName>;
</PackageDeclaration>
<ImportDeclaration>
import
<QualifiedName>javax.print.DocFlavor
</QualifiedName>;
</ImportDeclaration>
<TypeDeclaration>
<ClassOrInterfaceModifier>
public
</ClassOrInterfaceModifier>
<ClassDeclaration>
class
test
<ClassBody>{
<ClassBodyDeclaration>
<Modifier>
<ClassOrInterfaceModifier>
public
</ClassOrInterfaceModifier>
```

3-2-5 تحويل ملف XML إلى RDF

إن هدفنا في هذا البحث هو الوصول لصيغة موحدة لتمثيل كود المصدر لنتمكن من تطبيق منطق الويب الدلالي في الاستدلال للقيام بالتحليل الستاتيكي للنص البرمجي، لذلك سنقوم بالتحويل إلى إطار توصيف المصادر RDF، وتعتبر عملية التحويل بين XML وRDF أمراً هاماً ولكنه ليس بسيطاً، ولذلك توجهت العديد من الجهود ضمن W3C لدراسته. حيث قامت كلاً من التنظيمات التالية: GRDDL[1] و SAWSDL[2] بتقديم حلول

من أجل عملية التحويل التي أطلق عليها الرفع والخفض(lifting and lowering)، والتي تعتمد على تحويلات XSL أو ما يسمى ب XSLT بشكل مطلق وهو أمر أثبت عدم فعاليته بشكل كامل[10]، لذلك تم فيما بعد وتحديداً عام 2007 الاتجاه نحو طرق

أخرى مثل الاعتماد علىXQuery وهو ما سنقوم به في بحثنا، إنXQuery هي لغة استعلام وظيفية يمكن استخدامها للبحث واستخراج العناصر والخصائص من ملفات XML، وقد أصبحت من توصيات W3C منذ عام 2007، وتتميز بكونها سهلة التعلم خاصة عند وجود معرفة سابقة ب SQL، وهي مناسبة أكثر من XSLT عندما نتعامل مع البيانات التي يتوافر بينها علاقات، وسنستخدمها في بحثنا ضمن خوارزمية التحويل.

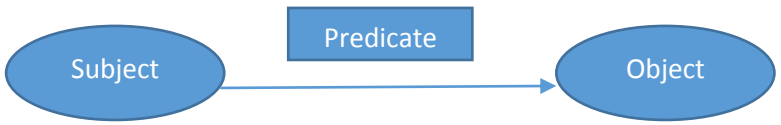
5-2-4 خوارزمية التحويل من XML إلى RDF:

تعتمد الخوارزمية على التنقل على بنية ملف XML الشجرية بطريقة أعلى-لأسفل Top-Down بهدف تحويل مستند xml إلى RDF\XML كصيغة مبسطة قابلة للسلسلة من RDF تحتوي فقط على وسوم ومحتوى نصي بدون خصائص، ولتحقيق ذلك يوجد مجموعة من الاعتبارات وضعت بالاستفادة من دراسات سابقة للصيغ المبسطة من RDF من قبل Melnik[9]:

- هدفنا الوصول إلى جملRDF حيث كل جملة هي ثلاثية S-P-O الشكل(7).
- كل وسم XML يعتبر predicate ب RDF ويتم تمثيله ك Resource له URI.
- يتم استخدام عقد وهمية(Blank node) ويعبر عنها ك Resource ليس له URI وتسمى بالشكل _:Nn حيث N أي حرف و n رقم العقدة الشكل(8).

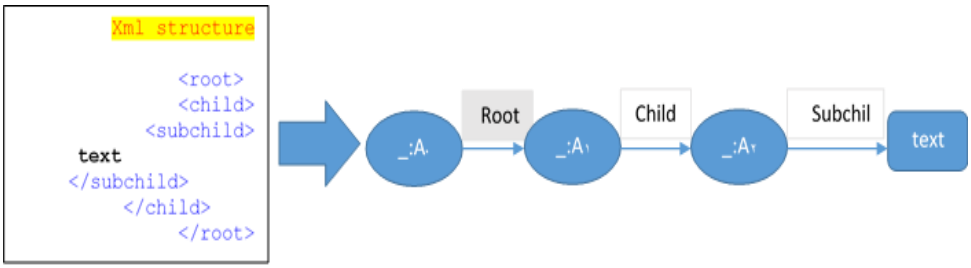
1.https://www.w3.org/TR/2004/NOTE-grddl-20040413/

2.https://www.w3.org/TR/sawsdl/



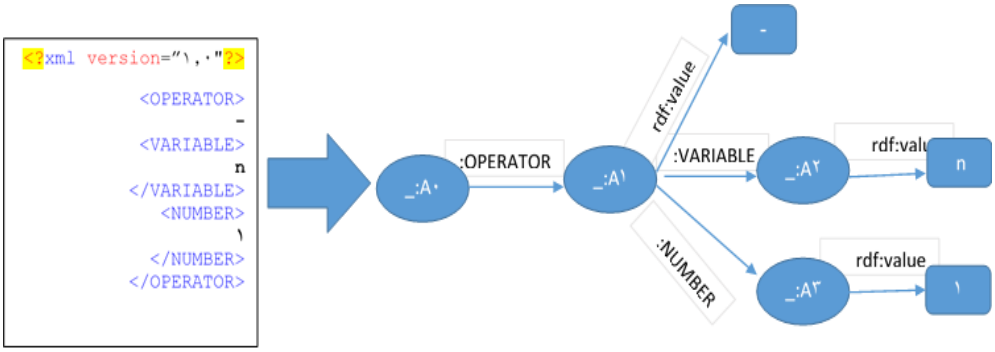
الشكل(7) جملة RDF

- بنية XML الناتجة عن تحويل شجرة التحليل لشيفرة المصدر لا تحوي عقداً جميع أبنائها من النوع وسم فقط، بل يمكن أن يكون أبناء العقدة الفرعيين وسوماً أو محتوى نصياً، كما في الشكل(9) فإن العقدة <Operator> لها ابن من نوع محتوى نصي وهو"-" وابنين آخرين <variable> و <Number>، في هذه الحالة لا يمكن اعتبار النص ك predicate وبما أنه ليس له أبناء بالتالي لا يتوفر object لتشكيل جملةRDF كاملة، فقد تم اعتبار النص ك object ومن الواجب إيجاد Predicate لتكوين الثلاثية المطلوبة، لذلك فقد قامت 1W3C عام 2004 بتعريف ال "value" وقد تم اعتبارها "كخاصية اصطلاحية للقيم المهيكلة ".



الشكل(8) بنية XMLوالبيان المقابل لها ب RDF وفقاً لنموذج Melnik

/.org/TR/rdf-schema31.https://www.w



الشكل(9) تمثيل بنية XML باستخدام بيان RDF

ويشير الرمز ":" الوارد في البيان إلى فضاء العناوين الذي نستخدمه، وفيما يلي سندرج خطوات خوارزمية التحويل:

- يتم زيارة عقد ملف ال XML وفق تابع عودي ونميز نوعين من العقد المعالجة، ولكل طريقة للتعامل معه:
  - Text عقدة نصية يتم إنشاء عقدة value مقابلة في ملف الrdf/xml.
  - Element عقدة وسم xml يتم إنشاء عقدة مصدر resource مقابلة.
- في حال كانت العقدة المزارة لها أبناء يتم استدعاء التابع العودي على الأبناء.

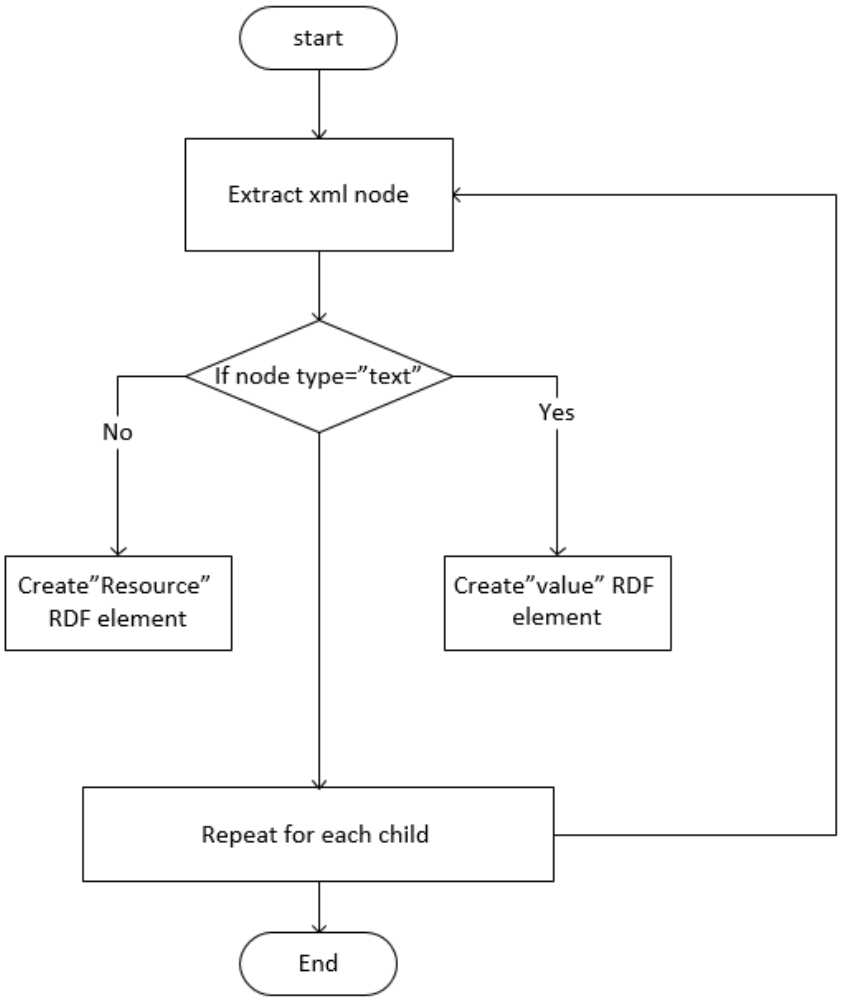
التابع الخاص بال XQuery الذي يحقق الخوارزمية يقوم بزيارة كل العقد ويستخدم التابعين element و attributes لإنشاء جمل ال RDF وفق التالي:

1. ينشئ عنصر XQuery وفق التابع element {computedname} {content}، حيث أن computedname عبارة عن QName وهو تابع يستخدم لإنشاء اسم العنصر مع إمكانية إضافة فضاء العناوين namespace للاسم المحلي للعنصر، فعلى سبيل المثال العنصر <operator>في الشكل(9) يكون الاسم المحلي له هو operator وبإضافة namespace الخاصة ببحثنا والتي هي http://noura.com/vulnerabilitydetection
   يصبح عنصر ال rdf\xml المقابل:
   <operator xmlns="http://noura.com/vulnerabilitydetection/" >،
   أما الـ content فهي كائن XQuery من النوع attribute (خاصية).
2. ينشئ خاصية وفق التابعattribute {computedname} {content}،

   حيث أن computedname هو rdf: parseType=" Resource" كما هو موضح في النتيجة المدرجة فيما يلي، أما ال content فتعتمد قيمتها على نوع العقدة الابن:

- إذا كانت عقدة وسم أي element فيتم زيارة أبنائها وإعادة الخطوة السابقة على كل ابن منهم.
- إذا كانت عقدة نصية فعندئذ سيتم بناء element جديد له ال predicate التالية rdf: value وسيشكل النص القسم object من جملة ال RDF، كمثال عليها العقدة "–" كيف تم تمثيلها في الشكل(9).

والشكل(10) يبين خوارزمية التحويل من XML إلى RDF.



الشكل(10) خوارزمية تحويل مستند XML إلى RDF

وبتطبيق التابع الذي يحقق الخوارزمية على ملف XML الوارد في الشكل(5) نحصل على النتيجة المبينة فيما يلي:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <rdf:Description>
      <OPERATOR xmlns="http://noura.com/vulnerabilitydetection/"
rdf:parseType="Resource">
         <rdf:value>-</rdf:value>
         <VARIABLE rdf:parseType="Resource">
            <rdf:value>n</rdf:value>
         </VARIABLE>
         <NUMBER rdf:parseType="Resource">
            <rdf:value>1</rdf:value>
         </NUMBER>
      </OPERATOR>
   </rdf:Description>
</rdf:RDF>
```

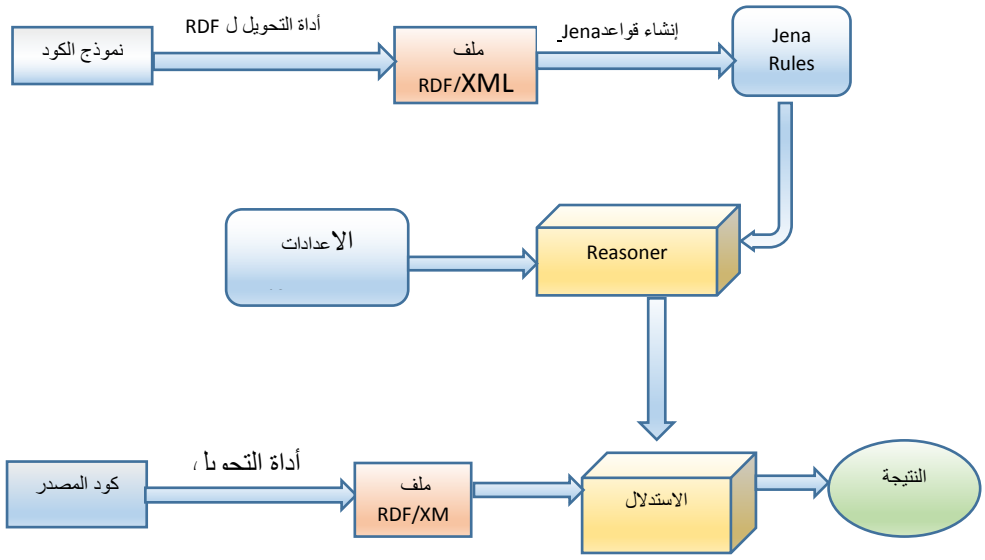
6. توظيف الأداة المقترحة في مجال التحليل الستاتيكي باستخدام تقنيات الويب الدلالي:

وفقاً لإحصائية NIST[1] _المعهد الوطني للمعايير والتكنولوجيا، وهو منظمة حكومية تجمع البيانات حول الثغرات في الويب عبر السنوات وتقوم بتخزينها وعمل إحصائيات لها_ فإن معظم الهجمات التي تستهدف تطبيقات الويب مؤخراً تستغل الثغرات الناتجة عن عدم ضبط تدفق البيانات عبر التطبيق أو بشكل أدق بسبب ضعف التحقق من بيانات الدخل للتطبيق IV، والتي تسبب مخاطر مثلXSS،SQLI .

لذلك فإنه من الضروري التحقق من بيانات الدخل للتطبيق وتحديد أي منها يشكل مشتبهاً به، ويمكن القيام بتحديد نقاط الدخول باستخدام نماذج الكود التي تشكل ثغرات يمكن استغلالها وتحقيق الهجوم من خلالها، وهنا يمكننا استخدام أداة استخراج حقائق المصدر التي تم تحقيقها في هذا البحث والتي تسمح بتحويل شيفرة التطبيق إلى توصيف مستقل عن اللغة متمثلاً بالصيغة RDF، والتي يمكن تطبيق تقنيات الويب الدلالي عليها.

1. https://www.nist.gov/

ومن ثم صياغة القواعد المقابلة inference rules للنماذج المختارة، ليصار إلى استخدام هذه القواعد إضافة لبعض الإعدادات الخاصة بمحرك الاستدلال الأمامي الخاص ب Jena في بناء reasoner دلالي، ومن ثم تتم عملية الاستدلال لتحليل النص البرمجي للتطبيق ومعرفة إذا كان يحوي نقاط دخول قد تشكل خطراً يمكن استغلاله لإحداث ضررٍ به والشكل[11] يوضح المنهجية المذكورة.



الشكل[11] منهجية التحليل المعتمدة على الاستدلال لكشف نقاط الدخول

ويعرف الاستدلال على أنه العملية المجردة لاكتشاف علاقات جديدة، أما ال reasoner[1]فهو الجزء من الكود الذي يقوم بهذه العملية.

1.https://www.w3.org/RDF/FAQ

ووفقاً ل jena حيث يتم نمذجة البيانات كمجموعة من العلاقات بين المصادر، فإن الاستدلال يعبر عن الإجرائيات المؤتمتة القادرة على إنتاج علاقات جديدة معتمدة على البيانات إضافة لمجموعة من المعلومات الإضافية بصيغة وجوديات أو علاقات، وهذه العلاقات الجديدة يمكن إضافتها لمخزن البيانات أو إرجاعها كنتيجة للاستعلام، وفيما يلي نبين الشكل العام لصياغة قواعد جينا:

```
Rule      := bare-rule .

             or [ bare-rule ]
             or [ ruleName : bare-rule ]
bare-rule := term, ... term -> hterm, ... hterm // forward rule
             or bhterm <- term, ... term // backward rule
hterm     := term
          or [ bare-rule ]
term      := (node, node, node) // triple pattern
          or (node, node, functor) // extended triple pattern
          or builtin(node, ... node) // invoke procedural primitive
bhterm    := (node, node, node) // triple pattern
functor   := functorName(node, ... node) // structured literal
node      := uri-ref // e.g. http://foo.com/eg
          or prefix:localname // e.g. rdf:type
          or <uri-ref> // e.g. <myscheme:myuri>
          or ?varname // variable
          or 'a literal' // a plain string literal
          or 'lex'^^typeURI // a typed literal
          or number // e.g. 42 or 25.5
```

حيث تتألف قواعد Jena من قسمين: عبارة if التي تشكل القسم الأول، وعبارة then التي تشكل القسم الثاني. وبما أننا سنستخدم المحرك الأمامي، فإنه سيبدأ ببيانات متاحة

ويستنتج بيانات جديدة عبر البحث بين العبارات حتى يجد تطابقاً للعبارة if، وعند حصول التطابق أي استيفاء الشرط يستنتج الجزء then.

وعلى سبيل المثال لو أردنا فحص الكود مقابل نموذج كود نقطة الثغرة المحتملة

```
/* Read data using an outbound tcp connection */
Socket sock = new Socket ("host.example.org", 39544);
```

JavaSocket والتي حصلنا عليها من موقع NIST كما يلي:

فإنه ووفقاً لمنهجيتنا سنقوم بتحويله لصيغةٌ RDF لنحصل على ملف نبين فيما يلي جزءاً منه:

وانطلاقاً من هذه الصيغة وعملاً بالشكل العام لصياغة قواعد جينا واعتماداً على

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <rdf:Description>
.
.
<VariableDeclaratorId rdf:parseType="Resource">
          <rdf:value>sock</rdf:value>
        </VariableDeclaratorId>
        <rdf:value>=</rdf:value>
       <VariableInitializer rdf:parseType="Resource">
       <Expression rdf:parseType="Resource">
        <rdf:value>new</rdf:value>
        <Creator rdf:parseType="Resource">
        <CreatedName rdf:parseType="Resource">
          <rdf:value>Socket</rdf:value>
        </CreatedName>
        <ClassCreatorRest rdf:parseType="Resource">
          <Arguments rdf:parseType="Resource">
            <rdf:value>(</rdf:value>
            <ExpressionList rdf:parseType="Resource">
               <Expression rdf:parseType="Resource">
                  <Primary rdf:parseType="Resource">
                    <Literal rdf:parseType="Resource">
                       <rdf:value>"host.example.org"</rdf:value>
```

مصفوفات الWildcards وال variables التي قمنا بعملها يدوياً والتي تشكل متحولات

```
@prefix vd: http://noura.com/vulnerabilitydetection /
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
[rule:
(?A744806 vd:CLASS_CONSTRUCTOR_CALL ?A744807 ),
(?A744807 vd:ARGUMENT_LIST ?A744808 ),
(?A744808 vd:EXPR ?A744809 ),
(?A744810 rdf:value ?wildcard56 ),
(?A744809 vd:DECIMAL_LITERAL ?A744810 ),
(?A744808 vd:EXPR ?A744811 ),
(?A744812 rdf:value ?wildcard77 ),
(?A744811 vd:STRING_LITERAL ?A744812 ),
(?A744808 rdf:value "ARGUMENT_LIST" ),
(?A744807 vd:QUALIFIED_TYPE_IDENT ?A744813 ),
(?A744814 rdf:value "Socket" ),
(?A744813 vd:IDENT ?A744814 ),
(?A744815 rdf:value ?entry_point ),
(?A744806 vd:IDENT ?A744815 ),
(?A744806 rdf:value "=" ) ->
(vd:unit vd:socket_entry_point ?entry_point)]
```

وثوابت النموذج يمكن صياغة قواعد الJena المقابلة للنموذج وفق مايلي:

وفي هذه المرحلة نستطيع تطبيق الاستدلال الأمامي عبر بناء reasoner دلالي باستخدام القواعد الناتجة عن المرحلة السابقة، إضافة إلى بعض الإعدادات الإضافية المتمثلة في نوع محرك الاستدلال المستخدم وهو النوع الأمامي مع خوارزمية RETE، وذلك بهدف استنتاج جمل جديدة في حال أن محرك الاستدلال وجد مطابقة للنموذج ضمن الكود المدروس الذي اخترناه وبالتالي فإن الكود يحوي نقطة دخول يجب معالجتها حتى لا تشكل خطراً على الكود، وفيما يلي نبين عملية إنشاء ال Reasoner:

```
Model m = ModelFactory.createDefaultModel();

Resource configuration = m.createResource();
configuration.addProperty(ReasonerVocabulary.PROPruleMode,
"forwardRETE");
configuration.addProperty(ReasonerVocabulary.PROPruleSet,
rules_filename);
// Create an instance of such a reasoner
Reasoner reasoner = GenericRuleReasonerFactory.
theInstance().create(configuration);
```

فلو حصلنا بعد تطبيق الاستدلال الخاص باختبار نموذج نقطة الدخول Javasocket على جمل(ثلاثيات) مثل الجملة:

(vd:unit vd:socket_entry_point “sock”)

عندها يتم وضع المتحول sock كمشتبه به ويعتبر مصدراً للخطر يجب التعامل معه.

وبتطبيق هذه المنهجية باستخدام كود يحوي نقطة الدخول المدروسة يمكن الحصول على رسالة توضح النتيجة كما يلي:

```
Inferred statement: [http:// noura.com/vulnerabilitydetection/unit,

http:// noura.com/vulnerabilitydetection/socket_entry_point, "sock"]

Taint variable: sock
```

ولا بد من الإشارة إلى الاعتبارات التالية:

- يبين الجدول(1) نتائج تجريبية نلاحظ من خلالها أن الزمن اللازم للانتقال من شيفرة المصدر مروراً بمرحلة الشجرة ثم XML وصولاً لمستند RDF يكاد يكون واحداً عند حجوم مختلفة في ملفات الجافا ومن الممكن أن يتأثر بشكل طفيف عند وجود بنى معطيات أعقد أو عند وجود تعليمات فتح ملفات والقراءة منها، أما في حالة ملفات C لاحظنا أن الزمن أصبح يزداد بشكل طردي بزيادة حجوم الملفات ولكن بنسب صغيرة ولم يتأثر بنمط المعطيات المستخدم بشكل كبير.

جدول(1) نتائج تطبيق الأداة على ملفات بحجوم مختلفة بلغتي الجافا و C

- يوجد أدوات تقوم بعملية استخلاص حقائق المصدر لكنها تعاني من بعض المحدودية، وسندرج فيما يلي أمثلة عنها موضحين النقاط التي تتفوق بها الأداة المقدمة ضمن البحث.

Java File	size	Time	C File	size	time
listSet.java	176 line	4 sec	Strings.c	15 lines	2 sec
Findtwo.java	30 lines	3 sec	Arrays.c	164 lines	3 sec
FileDisp.java	36 lines	3 sec	Testlong.c	380 lines	3 sec
Connt4.java	320 line	4 sec	Longer.c	717 lines	5 sec
saxAvg.java	450 line	4 sec	Files.c	200 lines	4 sec
Applet.java	600 line	4 sec	Fileslong.c	1000 lines	10 sec

الأداة SeCold (Source code Ecosystem Linked Data)

تقوم هذه الأداة باستخلاص حقائق كود المصدر القابلة للفهم من قبل الإنسان والآلة، وتسمح بتبادل وتشارك الحقائق الممثلة بصيغة قياسية واعتبرت نواة لأول مخزن linked data، متيحة إمكانية الوصول لـ 1.5 بليون ثلاثية[5]. و يمكن استخدام SeCold لأغراض البحث والتحليل والتنقيب في المعطيات، مشكلتها تكمن في الحبيبية التي تقدمها، حيث لا يمكن عبر هذه الأداة تقسيم سطر الكود لأكثر من حد معين بل يتم التعامل معه كوحدة متكاملة وهذا لا يساعد في التحكم بتدفق المعطيات والذي يحتاج إلى تفصيل أكثر من ذلك، إضافة إلى أن هذه الأداة خاصة بالجافا ولايمكن تطبيقها على تطبيقات مكتوبة بلغات أخرى، بينما الأداة التي قدمناها استطعنا تطبيقها على عدة لغات كما هو وارد في الجدول(1).

أداة Ganapathy & Sagayaraj

وهي أداة تستخرج البيانات الوصفية(metadata) من شيفرة المصدر باستخدام مولد الكود QDox الذي يستخلص تعاريف الصفوف والواجهات والطرق من ملفات الجافا، وتقوم بتخزينها بصيغة OWL عبر مكتبة ال Jena، وهكذا فإنها تخزن حقائق عالية المستوى عن الكود كالتعريفات متيحة بذلك مبدأ إعادة الاستخدام، في حين أن الكود الفعلي يتم تخزينه في مخازن أخرى مثل HDFS (Hadoop Distributed File System) ولذلك لا يمكن تطبيق تحليل الكود على الحقائق المستخلصة بهذه الأداة، وهنا تكمن أهمية الأداة التي قدمناها والتي تتيح لنا التمثيل للكود وفق الحبيبية المطلوبة دون وجود عائق اللغة.

نتائج وآفاق مستقبلية:

1- يقدم البحث أداة تقوم توليد شجرة التحليل CST التي تمثل النص البرمجي لتطبيق الويب المكتوب بلغة برمجة متقدمة ثم تحويلها إلى الصيغة الوسيطة XML،

وصولاً إلى لغة المصادر الموحدة RDF، وذلك انطلاقاً من ملف قواعد اللغة وشيفرة مصدر التطبيق.

2- تشكل ملفات RDF الناتجة تمثيلاً مناسباً من حيث الاستقلال عن لغة المصدر، ودرجة الحبيبية اللازمة لاستخدامها في عملية التحليل الستاتيكي للتطبيق كما لاحظنا في حالة JavaSocket فقد تم تقسيم سطر الكود لعدة وحدات مما مكننا من تطبيق منهجية تحليل مناسبة.

3- يمكن الاستفادة من هذه الأداة في مجال التحليل الستاتيكي لتطبيق الويب عبر وضع نماذج من الكود تشكل ثغرات أمنية، وصياغة القواعد المقابلة لها بعد تمثيلها وفق الصيغة الموحدة التي توفرها الأداة، مما يتيح فيما بعد إمكانية استخدام تقنيات الويب الدلالي عبر الاستدلال وفق هذه القواعد، وبالتالي الكشف عن الثغرات الأمنية.

4- يمكن العمل على تحسين الخوارزميات المستخدمة في صنع الأداة المقدمة بهدف تسريع عملية التحويل.

5- كما يمكن تضمين عدد أكبر من ملفات قواعد اللغات وإجراء بعض التعديلات لتضمين هذه اللغات مما يضمن مرونة الأداة مع تقديم تطبيقات بلغات جديدة.

المراجع:

[1] MYERS Andrew,2014- JFlow: Practical Mostly-Static Information Flow Control, Proceedings of the 26th ACM Symposium on Principles of Programming Languages (POPL '99), San Antonio, Texas, USA,p11.

[2] GERMAN Andy,2003-Software Static Code Analysis, CROSSTALK The Journal of Defense Software Engineering,p5.

[3] THURAISINGHAM Bhavani, 2013 - Security Issues for the Semantic Web, Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03) 0730-3157/03 $ 17.00 © IEEE

[4] GANAPATHY Gopinath, S. Sagayaraj, 2011-To Generate the Ontology from Java Source Code OWL Creation, (IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 2, No.2.

[5] IMAN Keivanloo, Christopher Forbes, Juergen Rilling , Philippe Charland ,2015,Towards Sharing Source Code Facts Using Linked Data, 22 p.

[6]5JENS Krinke ,Information Flow Control and Taint Analysis with Dependence Graphs, Published in the Proceedings Third International Workshop on Code Based Software Security Assessments (CoBaSSA),2007 in Vancouver, Canada,p6.

[7] MOHD. Ehmer Khan,2011-Different Approaches to White Box Testing Technique for Finding Errors, International Journal of software Engineering and Its Applications Vol. 5 No. 3.

[8] Dr. Bhavani Thuraisingham,2010- Security Issues for the Semantic Web,Program Director-Data and Applications Security-The National Science Foundation-Arlington, VA-On leave from The MITRE Corporation-Bedford, MA-Authorized licensed use

limited to: Univ of Texas at Dallas. at 17:34:12 UTC from IEEE Xplore

[9] DECKERr Stefan,Sergey Menlink- Frank Van Harmelen, Dieter Fensel, Michel Klein, Jeen Broextra- Micheal Erdmann, Ian Horrocks,2015- The Semantic Web:The Roles of XML and RDF 1089-7801/ 00/$10.00 ©2015 IEEE.

[10] AKHTAR Waseem, Jacek Kopeck´y, Thomas Krennwallner, and Axel Polleres,2008- XSPARQL: Traveling between the XML and RDF worlds and avoiding the XSLT pilgrimage, Digital Enterprise Research Institute, National University of Ireland, Galway-STI Innsbruck, University of Innsbruck, Austria Proceedings of the 5th European Semantic Web Conference (ESWC2008),
Springer, 2008

[11] Lian Yu, Shi-Zhong Wu, Tao Guo, Guo-Wei Dong, Cheng-Cheng Wan, and Yin-Hang Jing, 2011- Ontology Model-Based Static Analysis of Security Vulnerabilities, Springer-Verlag Berlin Heidelberg 2011

[12] KAHANI Mohsen, Ala EkramiFard, 2015 -Providing a Source Code Security Analysis Model Using Semantic Web Techniques, Second International Congress on Technology, Communication and Knowledge (ICTCK 2015) November, 11-12, 2015 - Mashhad Branch, Islamic Azad University, Mashhad, Iran

[13]Terence Parr, 2013-The Definitive ANTLR 4 Reference, Dallas, Texas • Raleigh, North Carolina, 283 p.

[14] Auniversal security infrastructure for ISPs and corporate networks using NFV-enabled technologies: the SHIELD project, SHIELD Newsletter n.4 - March 2019 Mar 22, 2019.

[15] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, Jérôme Siméon, 2007- XQuery 1.0: An XML Query Language W3C Recommendation 23 January 2007.

[16] Ian Dickinson, Andy Seaborne, Chris Dollin, Kevin Wilkinson, 2004- Jena: Implementing the Semantic Web Recommendations, Jeremy J. Carroll,Dave Reynolds HP Labs, Bristol-UK-- HP Labs, Palo Alto-CA. USA.