Defence Research and Development Canada
Recherche et développement pour la défense Canada

Security Posture Assessment Demonstrator and Experimenter

Application Tools to Support Security Research Testing

Eugen Bacic, Glen Henderson and Larry Tremblay

Defence R&D Canada – Ottawa

Canada

Contract Report
DRDC Ottawa CR 2011-011
May 2011

Security Posture Assessment Demonstrator and Experimenter

Application Tools to Support Security Research Testing

Eugen Bacic
Bell Canada

Glen Henderson
Bell Canada

Larry Tremblay
Bell Canada

Prepared by:

Bell Canada
160 Elgin St., 17th Floor, Ottawa, Ontario, K2P 2C4

Project Manager: Reginald Sawilla
Contract Number: W7714-071028
Contract Scientific Authority: Reginald Sawilla

Defence R&D Canada – Ottawa

Contract Report

DRDC Ottawa CR 2011-011

May 2011

Scientific Authority

Original signed by Reginald Sawilla

Reginald Sawilla

Approved by

Original signed by for Julie Lefebvre

Julie Lefebvre
Section Head/NIO Section

Approved for release by

Original signed by for Chris McMillan

Chris McMillan
Head/Document Review Panel

Abstract

There is a recognized need within the network defence community for analysis tools that can assist in the evaluation of networks for vulnerabilities and aid in the design of robust networks. This paper describes the initial design and development work to create a unified framework that can leverage research, commercial and academic security analysis tools in support of ongoing network defence analysis practices. An initial examination of network modeling and model manipulation techniques is provided, including: appropriate data representation, automated and interactive interfaces and necessary modeling capabilities. The analysis portion of the proposed framework is described both in terms of its overall pluggable architecture and in terms of sample modules that could be integrated within and leveraged by the overall analytical framework. The data storage mechanisms for model data is presented, not only for data generated through the analysis process, but also the means by which external data sources can be leveraged to provide near real time contextual data for the various analytical modules. To illustrate the framework in action, data flow and transformation for the use of a specific module, the MulVAL analysis engine, is described. A series of network models were tested against the framework and sample module showing the validity of the design approach.

Résumé

Il est reconnu que la collectivité de la défense de réseaux a besoin d'outils d'analyse pouvant aider à évaluer les vulnérabilités des réseaux et à concevoir des réseaux solides. Cette étude fait état des premiers travaux de conception et de développement visant à créer un cadre unifié pouvant mettre à profit les outils de recherche, commerciaux et universitaires d'analyse de la sécurité à l'appui des pratiques d'analyse de la défense de réseaux. Un premier examen des techniques de modélisation de réseaux et de manipulation de modèles est présenté, y compris une représentation appropriée des données, certaines interfaces automatisées et interactives et les capacités de modélisation nécessaires. La partie analytique du cadre proposé est décrite aussi bien du point de vue général de son architecture enfichable que du point de vue des modèles de modules qui pourraient être intégrés au cadre global d'analyse et en profiter. Les mécanismes de stockage des données du modèle sont présentés non seulement aux fins des données générées durant le processus d'analyse, mais aussi afin d'établir les moyens par lesquels on peut exploiter des sources extérieures de données pour obtenir des données contextuelles en temps quasi réel à l'intention des divers modules d'analyse. Afin d'illustrer le cadre en action, le flux de données et la transformation en vue de l'utilisation d'un module donné, on présente une description du moteur d'analyse MulVAL. Une série de modèles de réseaux vérifiés par rapport au cadre et au module modèle témoignent de la validité de l'approche conceptuelle.

This page intentionally left blank.

Table of contents

1 Introduction

This document describes the results from research and application development activities associated with the creation of a set of analysis tools and capabilities to support defence research regarding network security posture. This report is the third in a series that builds upon previous work that:

- identified and prioritized a set of requirements for this analysis tool set; and
- developed a series of network models which will be used to demonstrate and test the analysis tool set.

As part of the original requirements gathering process, a work plan was developed that identified the development goals in terms of a series of work tracks. These tracks were defined in the following manner:

1. Network Model Manipulation Tool Set Development
2. Analysis Tools Set Development
3. Model Storage and Representation
4. Model Building and Testing

The work documented in this report is presented in separate sections for each of these four work tracks. Where appropriate, the design decisions, lessons learned and the recommendations for future work are documented with the work track item to which they apply.

This report is meant to serve as a design and development compendium regarding the development work that has been done to date on the analytical tool set. The content of this report will serve as input to the final development report which will be submitted with the completed tool set.

2 Network Model Manipulation

The Network Modeller application is a standalone program used to visually create networks for use with the MulVAL tool. Network Modeller represents its modelled networks in an XML format which can be transformed into a set of Datalog predicates used by MulVAL for analysis of the network in order to find attack vectors against the network and assess the network's defensive posture.

2.1 Design

The Network Modeller application is built using the Java programming language. Its basic design is split into three parts:

- XML representation
- Data representation
- Graphical display

Each of these elements interacts with the others in order to provide a complete representation of the network model as a document that is loaded and saved by Network Modeller.

2.1.1 XML Representation

Network Modeller models are made persistent objects in an XML format in storage. An XML schema to describe network models has been devised, and is detailed in *Network Scenarios Architecture for Security Research Testing Report (Annex A: XML Representation)*. For the Network Modeller application, a tool was required for reading XML network representations and facilitating their expression in data. To that end, the JDOM library has been used for its modelling and expression capabilities. When opened, a network representation is read by JDOM as a collection of Java objects representing each XML element. These objects are represented in the same fashion as the XML document, with objects that represent elements stored in collections under their superior elements within the object hierarchy.

2.1.2 Data Representation

The data representation of a network model mirrors the hierarchical nature of the XML representation. That is, a model has a single root object that contains a list of the sites in the model. Each site contains a list of the zones within it, each zone contains a list of its machines, and so on. Using the object model presented by the JDOM library, each data object contains a reference to its JDOM object. This allows changes made to data objects

to be immediately reflected in the JDOM object, which in turn are reflected in the JDOM XML document.

The hierarchical nature of the network model data is expressed in a tree view display in the Network Modeller application. This view allows users to see the nested nature of the model (e.g. machines contained in zones contained in sites.) It also provides the user with tools to examine the details of a given element, such as the operating system and version for a machine, as well as the ability to edit a given element to change its attributes. As noted, these changes are immediately reflected in the XML objects corresponding to the element's data object, and so are changed in the XML document.

2.1.3 Graphical Display

The graphical display for Network Modeller uses the Java Universal Network/Graph (JUNG) framework as its graph representation. Objects that are displayed on the graph (sites, zones, machines) are painted onto the graph panel, which allows the user to zoom in and out, as well as pan the display to allow further-out broad views or close-in detail views. Each visual element is linked to its data element, allowing the graphing framework access to required visual elements for a given element, such as which icon to display and coordinate offsets within the graph for positioning. The visualization capability is slated for the next series of development activities is only partially researched at this time.

2.1.4 Other Features

The Network Modeller provides several "standard" features that are expected in an application such as cut and paste, as well as some convenience features to assist users in rapidly creating and manipulating network models.

SQL Model Persistence The Network Modeller application allows persistence (storage) of network models is through the use of XML files only. Interaction with SQL databases is limited to scripts (written in the Python programming language.) A SQL command file is provided for the creation of the database representation of network models in Microsoft SQL Server. Scripts are also provided for parsing a Network Modeller XML file and populating a database with the model, and for reading a database containing a model and writing out an XML file for the Network Modeller application to read.

Templating The templating function of Network Modeller allows users to create one instance of an element, and select that element to be written as a template. "An element" may be selected as a single zone or machine, but the template will include all sub-elements of the selected element when written as a template. In this fashion, a machine element can be created that reflects, say, a desktop image that consists of a particular operating system and suite of software and services that is deployed many times within the organization. After creating the machine and adding all appropriate sub-elements, the user can save that

machine as a template. Once saved, new instances of the template can be inserted into the model with a click on a menu. Templates can be made using any valid element in a network model, which can be as simple as a software package, or as complex as an entire site containing zones which are populated with machines.

Cut and Paste Working on the same principle as templating, cut and paste within the Network Modeller allows the user to make a copy of an existing element (including all sub-elements) in a model and create copies of it within the model. The difference between templating and copying is simply that templating writes the element to a file, which is persistent until deleted. A copied element remains in memory and is only available until a different element is copied.

2.2 Design Rationale

The design for Network Modeller was presented in the previous project report *System Description and Work Plan (Section 4: Demonstrator Architecture)*. As such, much of the design rationale has been explained and can be referenced in that document. Over the course of development, some aspects of the design have been changed to reflect new priorities or changed requirements. Those changes are discussed in this section.

2.2.1 XML Schema

Over the course of the design, the plan for storage of models has been changed. The path and motivations for these changes can be found in *Section 2.3:Lessons Learned*. Ultimately, these changes lead to a smaller and more efficient storage medium for network models. This makes conversion of stored models to and from other storage formats (other XML schema, databases, etc.) much simpler and manageable.

2.2.2 SQL Interaction

The specification for the Network Modeller requires the ability to store and retrieve models in SQL databases, but this is not directly supported by the application. As noted, all SQL interaction is performed via scripts that are dedicated to converting from XML format to SQL schema, and vice-versa. This design simplifies the Network Modeller application, as it does not require code to handle both XML and SQL storage. The use of small, dedicated scripts that perform only conversions shifts the work for altering stored models to directly editing XML files or manipulating SQL data directly via transact SQL statements. In both cases, using conversion scripts means that changes to the way that data is altered data do not require changes to the scripts, meaning that user actions and desires will be much less likely to require code changes. Extensions of the XML schema can then be easily reflected in the scripts with minimal effort, as the scripts simply require updating to handle new fields or tables for conversion.

2.2.3 Choice of Libraries

One of the requirements for the Network Modeller application is that it be created using existing tools and libraries, ideally open-source. The major decisions required for Network Modeller were the choice of graph and XML tools.

JUNG is a popular and very extensible framework for drawing graphs with Java. It is packaged with numerous predefined layout formats, and more importantly the means to define new layout methods, which will be necessary for Network Modeller. The ability to customize vertex and edge graphics, as well as almost all other aspects of generating, displaying, and interacting with graphs is also very important for Network Modeller. It is also the graphing solution used by RapidMiner, which is the tool used for the analytical portion of the solution. Given its shared role with other components of the solution and the extensibility to allow Network Modeller to create and handle displays in any manner, JUNG was an obvious choice from the beginning.

JDOM provides access to XML entities and structures in a Java-specific manner that rationalizes the hierarchical nature of XML representations with Java objects. Any change made to the Java object representing an XML entity is immediately reflected in the XML document loaded in memory. This synchronization between the XML document and the Java objects is seamless and transparent to the developer, and the only step required by the developer is to write the changed document to storage after changes are made. By providing a simple and intuitive way to manipulate XML within with minimal development overhead, JDOM was easily the best choice for this application.

Other libraries in use are comprised of libraries that are required for JUNG or JDOM, and the log4j library which is a standard logging tool for Java.

2.3 Lessons Learned

The following sections detail significant design issues or lessons that have been learned during the investigation of the application tool set.

2.3.1 CIM

Initial design documentation for the Network Modeller application called for the use of the Common Information Model (CIM) XML schema. This schema provides a robust definition for modeling networks and systems in an XML format. It is, however, extremely large and defines significantly more detail than is required for this application. In addition, the sheer size of the CIM schema (some 1400+ individual schema definition files) made determining which subset of the schema would be required a very daunting and time-consuming task. Given this and the possibility of the JNDMS data format becoming

available for works distributable outside of DRDC, the decision was made to pursue storing the Network Modeller data within the JNDMS format.

2.3.2 JNDMS

Investigation of the JNDMS format led to several barriers to its use for the Network Modeller. First and foremost is that of the JNDMS model being a database schema, rather than an XML schema. While an XML format is easily and transparently extended to include the elements required for the Network Modeller application to include new elements, it is highly unlikely that extensions to the JNDMS database schema would have been accepted. This would necessitate a hybrid storage mechanism where the data that can be stored in a JNDMS database is stored there, and then a supplemental data storage is used for the data that cannot be stored in the JNDMS database.

Secondly, there is no mechanism built into JNDMS to export its own XML format available. This meant that a means to extract the data from the database and transform it into an XML format would need to be devised with the supplemental data also pulled out of whatever alternate storage mechanism and synthesized into the XML output.

2.3.3 Custom XML Format

It became clear that the path of desirability was to move back to a pure XML format. During the initial design stage, the CIM XML format was the forerunner for use with the Network Modeller. As has been discussed, its complexity is one reason for investigating JNDMS instead. Ultimately, it was decided that it is much more desirable to use a customized XML format for this application. As seen over the development process to date, using an already-defined standard reveals its own problems such as excessive complexity, poor suitability for the task, and requirement for extension to include all elements. In short, there is no standard XML format already defined that will suit this task perfectly, or without an extensive learning curve that will slow down adoption by third parties.

By defining a task-specific XML format tailored for Network Modeller, third parties are given a simple and straightforward format that Network Modeller wants. If someone is using CIM, then they can write XML transformations (XSLT) to convert their data into the much simpler format needed by Network Modeller. This can be done for any other XML format, whether it be a standard or proprietary, that any third party is using. With regards to JNDMS, an application can be easily written to extract the desired network data from a JNDMS database and from that write out a Network Modeller XML format file that can be read by the application. (The previously mentioned problems with missing data in the JNDMS schema are then shifted to the converter program to solve.) With Network Modeller being open-source, the code for an application that extracts data from a JNDMS database can be distributed to third parties without revealing the entire JNDMS schema

to them. This allows those third parties to use such a JNDMS data extracting tool as a template/example for creating their own version for whatever proprietary database-based storage they may be using. This has the net effect of decoupling the Network Modeller application from any data storage format expectations and complexities besides its own concise and compact format. Given a simple format that Network Modeller is expecting, third parties can easily create conversion tools given their presumably expert knowledge of their own data storage with minimal learning curve with regards to the Network Modeller format.

3 Analysis Tool

The open source RapidMiner application is used to create experiments that will invoke the MulVAL and AssetRank tools to analyze networks whether created by the Network Modeller application or other means.

3.1 Design

There are currently three analytical modules that have been produced for this project:

- MulVAL Module
- AssetRank Module
- Generic Module

The MulVAL and AssetRank analytical modules (called an *operator* in RapidMiner terminology) are each contained within their own package (a Java JAR file). In addition, there is a package of common elements that are shared between the analytical modules, which also contains the generic module.

3.1.1 MulVAL Module

The MulVAL module presents the MulVAL Invoker operator for RapidMiner. This is a specialized operator that presents the various options of the MulVAL tool for the user to set and modify. After specifying the input file containing the Prolog predicates defining the network and setting options and configuration items, the MulVAL operator is executed. Upon a successful run of MulVAL, the output files (arcs and vertices) are read, and those results placed in RapidMiner ExampleSets, a structure dedicated to holding arbitrary data. These ExampleSets, one for the MulVAL arcs, and the other for the vertices, are then passed to the next operator in the experiment.

To support the RapidMiner MulVAL operator, the original MulVAL executable, which was written as a UNIX shell script, was rewritten as a Python script. This rewriting allowed several goals to be achieved:

- Rewriting in the Python language removes a dependency on the UNIX shell, allowing MulVAL to be more easily used on other platforms, such as Microsoft Windows;
- A new option to specify a working directory allows the MulVAL created files (working files and output files) to be confined to a directory other than the directory containing the input file; and

- A new option to select the format of the created attack graph graphic from several options.

3.1.2 AssetRank Module

The AssetRank module presents the AssetRank Invoker operator for RapidMiner. This is a specialized operator that presents the various options of the AssetRank tool for the user to set and modify. Rather than specifying a file containing the input data as in the MulVAL operator, the AssetRank operator expects two ExampleSets, one containing arcs and the other vertices, as its inputs. That is, The AssetRank operator must have operator(s) before it in the experiment to gather its data. Upon a successful run of AssetRank, the output files that AssetRank creates (again, arcs and vertices) are read, and those results placed in ExampleSets which are then passed to the next operator in the experiment.

To support the RapidMiner AssetRank operator, an additional Python script was created to be called by the AssetRank operator. This is due to the architecture of the AssetRank tool, which expects a small ”driver” script to be created that will load the AssetRank tool itself, set any options, and then invoke AssetRank. This additional script acts as a ”dynamic driver” for AssetRank. When invoked, it accepts command line parameters that correspond to the options available to AssetRank. The AssetRank tool is loaded, options are set based on the parameters passed in, and then AssetRank is invoked. As with the rewritten MulVAL script, the AssetRank dynamic driver allows the user to specify a work directory in which all working and output files will be written.

3.1.3 Generic Operator

Unlike the MulVAL and AssetRank operators, the generic operator is meant to allow the user to execute any arbitrary command to act upon the data presented it. Two ExampleSets are expected as inputs from the previous operator in the experiment, and the user specifies file names to which they are to be written. It is expected that whatever command is invoked will produce a pair of output files, the names of which are also specified by the user in the generic operator’s configuration. Once the names of the input and output files are specified, the user specifies a command in a text input field. When the experiment is executed, the ExampleSets that are passed into the generic operator are written to the input files named by the user, then the command is executed verbatim, as if the user had issued it at a command prompt. After executing the command, the generic operator attempts to read the files given by the user as the output files, and any data read is put into ExampleSets. Assuming this is successful, the ExampleSets are then passed to the next operator in the experiment.

3.2 Design Rationale

The design for the analytical modules is presented in the *System Description and Work Plan Report*. Over the course of development, that design has been changed to reflect new priorities or changed requirements. Those changes are discussed in this section.

3.2.1 Model Analysis and Prioritization Operators

Initial design called for an ”Input Operator” which would read the files produced by the Network Modeller in order to validate the XML data and then transform the XML into a data form usable by the defined RapidMiner operators. After discussion amongst the project team, and with the agreement of the Project Authority, this functionality has been moved from being a RapidMiner operator to being a standalone script that performs the XML transformation (XSLT) of a Network Modeller output file into the proper format for the analysis tool to be used (MulVAL - Prolog predicates, AssetRank - CSV files, etc.). Note, however, that the script itself will be called from within the RapidMiner framework and the user experience will see a unified architecture and process for all modelling, transformational and analysis functions.

3.2.2 AssetRank and MulVAL Operators

Initial design called for these operators, which would be for invoking MulVAL and AssetRank respectively. For simplicity’s sake, these operators have been renamed to ”MulVAL Invoker” and ”AssetRank Invoker”

3.2.3 Pre-processing and Post-processing Operators

The initial design defines these operators, and several useful examples are are included with the RapidMiner suite. Future, more specific processing operators may be called for, and may be written as script files that can be invoked by the RapidMiner generic operator, or as Java plugin modules that will perform the desired processing directly within RapidMiner if sufficient development experience is available.

3.2.4 Visualization Operator

This operator, along with all other visualization tasks, is slated for the next series of development activities. However, the visualization operator is the mechanism through with visualization capabilities are linked to the analysis tool.

3.2.5 Export Operator

As with the pre-processing/post-processing operators, this functionality is implemented by RapidMiner operators for almost any format needed.

3.3 Lessons Learned

At this juncture, it is evident that the initial design for the RapidMiner portion of the project was generally correct. Over the course of development, the design of the individual operators underwent small iterative changes, but are substantively as designed. The MulVAL and AssetRank operators were generally implemented as planned, with only very minor changes to the planned design. Several utilities are implemented scripts that are invoked from within a RapidMiner experiment using the generic operator supplied by RapidMiner, rather than by creating additional operators in Java and packaging them for RapidMiner. This provides much simplification for users, allowing them to develop the scripts by whatever means they care to, in whatever language they care to, without requiring knowledge of Java programming and creation of RapidMiner operators, or again requiring external expertise for their development.

4 Data Storage and Modelling

This section describes the activities completed under the third work track, that is, the Data Storage and Modelling track. Specifically, the work performed under this track included the following activities:

1. Merging the project network model structure with the National Vulnerability Database (NVD) data feeds.
2. Extending the project network model to include MulVAL specific components.
3. Generation of MulVAL predicates from the project XML model and the NVD information feeds.

The work performed within this track integrates, leverages and builds upon some work elements presented earlier in this document and in the *Network Scenarios Architecture for Security Research Testing Report*. These work elements include:

- The Network Modeller application that leverages the *RapidMiner* framework
- The generic and MulVAL-specific module extensions
- The solution XML model to represent network elements (sites, zones, systems, services, accounts, software, etc...)

In essence, the work performed under this track has developed a software component, the MulVAL Code Generator, with the following capabilities:

1. It can integrate with the Analysis Tool via the MulVAL-specific analysis module.
2. It can generate MulVAL ready Datalog predicates from a given network model in the project XML format.
3. It can integrate information from NVD information feeds to create vulnerability related MulVAL predicates.
4. It ensures that the expressions transformed from XML to Datalog are syntactically correct.

4.1 Component Design

The MulVAL Code Generator is designed as per the following diagram.

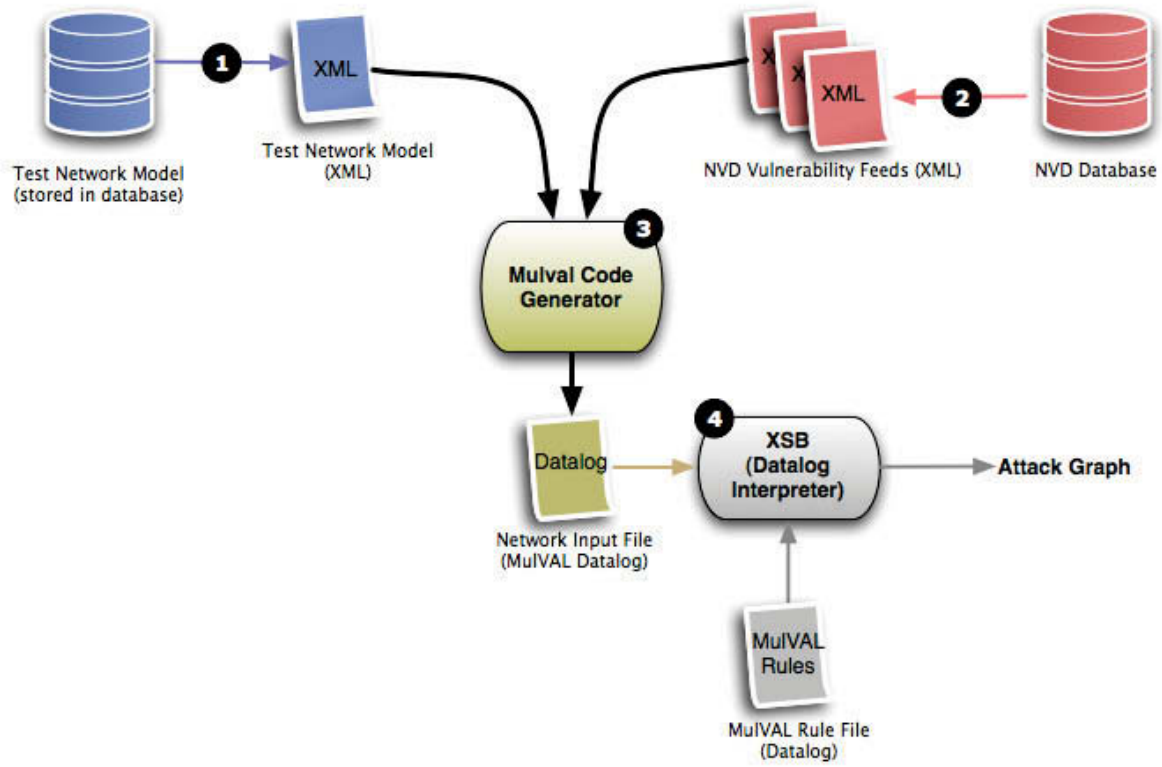

Figure 1: *MulVAL Code Generator Architecture*

In terms of the information flow within the architecture, the source is a network model (1) which is:

- designed via the network modeller software,
- stored in a database, and
- represented in the project network model XML schema.

This information source forms one input to the Code Generator. Externally to the software, the NVD host organization makes available a series of XML files which are extracts (2) from their vulnerability database. These extracts are provided as individual files, in the NVD 2.0 XML schema, with each file containing the vulnerabilities found in a specific year (e.g. *nvdcve-2.0-2009.xml* for the year 2009). These information files are the second set of inputs to the Generator process. The Generator combines these elements (3) and

creates MulVAL-specific Datalog that can be used as input to the MulVAL logic code (4) as executed through XSB to identify potential attack paths against the modelled network.

4.1.1 Design Principles

The Generator process was built using these principles as the prime design requirements:

1. The process must generate syntactically correct Datalog as per XSB's logic engine interpreter.
2. The process must generate predicates that are compliant with the MulVAL project's defined predicates.
3. The process must be able to leverage any or all NVD input files as part of the processing (multiple years).
4. The process must associate all NVD specified vulnerabilities with all applicable software in the input XML documents.
5. The process must create a separate Datalog file as output.
6. The process must complete in a reasonable length of time. (Note that specific performance metrics were not stated as requirements)
7. The process and code must be easy to easy to understand and modify in the face of new requirements.

The Generator process utilizes the following software components:

- libxml2 with Python bindings for XML parsing and searching
- Python 2.5.2 (r252:60911) built from the GCC (Gnu compiler) version 4.2.4
- Ubuntu Linux 8.04 (Hardy Heron) using the standard linux 2.6.24-25-generic kernel

It is significant to note that there are no software dependancies that would prevent the Generator process from being used under another operating system that supports the list of software components defined above.

4.2 Architecture

The solution is composed of three separate components:

- The *generateMulval* module which provides a front end interface to the generation process.
- The *mulvalWriter* module which generates the MulVAL Prolog statements.
- The *nvdLookup* module which interfaces with the NVD XML files.

These components are linked as per the following diagram.

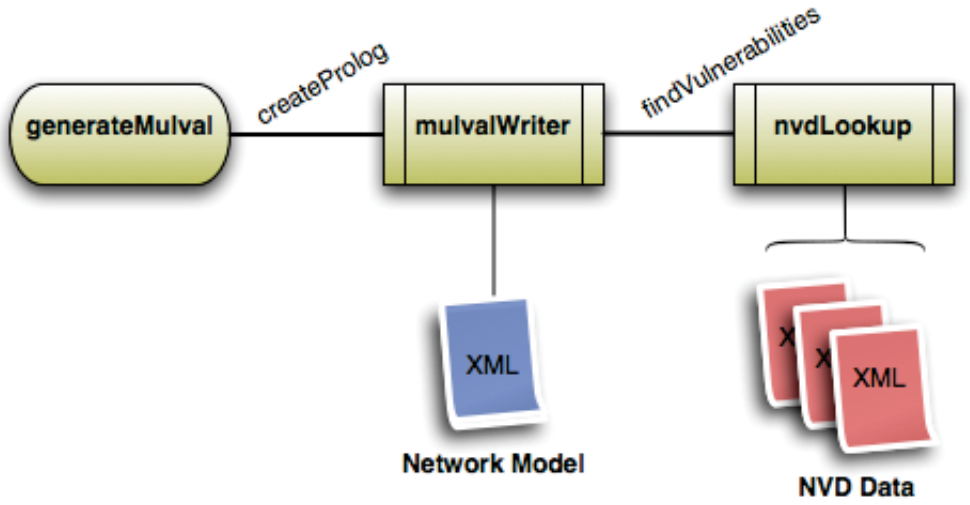

Figure 2: *Prolog Generator Class Map*

The *mulvalWriter* and *nvdLookup* Python objects are classes. The *generateMulval* module instantiates an instance of the *mulvalWriter* class and an *nvdLookup* object is instantiated by *mulvalWriter*.

The *generateMulval* Python module performs the following actions:

1. Parse the command line in the format: *generateMulval xml-input prolog-output nvd-directory*
2. Create a *mulvalWriter* object and supply the target input file name.
3. Instruct the *mulvalWriter* object to create the Datalog output at the designated output file name.

The *mulvalWriter* Python class, specifically the public method *createProlog*, performs the following actions:

1. Walk the XML structure and create MulVAL statements as appropriate.
2. When presented with a software component, lookup in the *nvdLookup* class to find any related vulnerabilities.
3. Acquire the properties of any vulnerabilities found in the network XML data.
4. Cache the vulnerability information related to any software/version component to reduce the number of lookups needed.

The *nvdLookup* Python class, specifically the public method *findVulnerabilities*, performs the following actions:

1. For each identified software component, search all NVD XML files to find the vulnerabilities associated with that software.
2. For all associated vulnerabilities, extract the relevant MulVAL data (e.g. range and impact).
3. Return a list of vulnerabilities and their properties.

4.3 MulVAL Predicates

MulVAL-ready code is created using the following general rules to generate MulVAL Datalog predicates from the source network model XML. These rules can be reviewed in the content of the XML schema defined in Annex A of the *Network Scenarios Architecture for Security Research Testing Report*.

MulVAL Predicate	Creation Rule
networkServiceInfo	The <service>tag includes attributes for port, protocol and account under which the service is run. The service tag provides the name of the software and the version tag provides the software version. When combined, those two data elements create the software identity that can be used to lookup information in the NVD. The parent tag holds the host machine data which uniquely identifies the machine on which the service is hosted.
hasAccount	The <account>tag includes attributes for the account name and type. The parent tag holds the host machine data which uniquely identifies the machine on which the account is located.

nfsExportInfo	The <share>tag includes attributes for the name, export and permissions on the share. The grandparent tag (share to file sharing service to machine) holds the host machine data which uniquely identifies the machine on which the share is located.
installed	The <software>tag includes attributes for the name of the software and the software version. When combined, those two data elements create the software identity that can be used to lookup information in the NVD. The parent tag holds the host machine data which uniquely identifies the machine on which the software is hosted.
inSubnet	The machine tag name attribute is combined with the parent zone tag name attribute to indicate that a machine belongs to a specific zone.
vulExists	After the lookup to the NVD files, each machine will be associated with a series of vulnerabilities. the vulExists predicate is created using the machine's unique name (including site name), the CVE id and the name of the software associated with the vulnerability.
vulProperty	For each vulnerability detected in the network, the properties for that vulnerability are added in a vulPropety predicate. This entry is composed of the CVE id, the impact and the range. See the section below which describes how these elements are determined.
hacl	hacl rules are created using the <acl>entry. See the note below for how this entry is formulated.

Table 1: MulVAL Creation Rules

4.3.1 Vulnerability Properties

A *vulProperty* predicate is created once for each vulnerability that is found in the modelled network. The base information for the *vulProperty* predicate is taken from the CVSS data set within the NVD 2.0 schema. These information elements are:

- cvss:access-vector which can take the values of LOCAL or REMOTE
- cvss:confidentiality-impact which can take the values of NONE, PARTIAL or COMPLETE
- cvss:integrity-impact which can take the values of NONE, PARTIAL or COMPLETE

- cvss:availability-impact which can take the values of NONE, PARTIAL or COMPLETE
- cvss:security-protection which can take the values of NONE, PARTIAL or COMPLETE

The rules for generating *vulProperty* are as follows:

1. If the access vector is LOCAL, the range value of *vulProperty* is 'localExploit', otherwise it is 'remoteExploit'.
2. If the availability-impact is not NONE the *vulProperty* impact value is 'dos'.
3. If the confidentiality-impact or integrity-impact values are not NONE the impact value is 'privEscalation'.
4. If the security-protection tag is present the impact value is 'privEscalation' .
5. the impact value is 'none' by default.

4.3.2 Subnet-based Access Control Rules

The *hacl* predicate leverages the use of subnets. As defined in the table above, it is possible to state that a specific machine is in a particular subnet using the *inSubnet* declaration. Thus if machine M1 is in subnet S1, the rule: *inSubnet(M1,S1)* would instantiate that relationship. The hacl rules, therefore, leverage this information to create subnet-based access control rules. If it is possible for any machine in subnet 1 to reach subnet 2 over http (port 80), the following *hacl* rule would express this fact.

```
hacl(M1,M2,http,80) :- inSubnet(M1,subnet1), inSubnet(M2,subnet2).
```

Specifically, this rule states that any host in subnet 1 can reach any host in subnet2 over http on port 80. All zone-level rules are generated using this Datalog expression format.

4.4 Design Rationale

The choice of Python for the development language allows us to meet many of the previously stated design principles. Specifically, Python code is easy to understand and modify and therefore, is the best approach for trying to keep project network XML, NVD XML and MulVAL Datalog data formats synchronized. The Python code leverages the Python bindings to the *libxml2* XML C parser. There are several choices for the XML parsing component to be leveraged by the Generator, including some potentially superior candidates like *lxml*, and *4suite*. There are also a number of less promising candidates like the PyXML project (appears to be dead) and the native XML parser utility that is included

with Ubuntu Linux 8.04 (deemed not sufficiently robust enough to meet project needs). The *libxml2* bindings were chosen for the generator because it was believed that the C-based code would provide the best performance, although an empirical study of both libraries would validate this position.

The use of Python classes supports the reuse of objects in other contexts. For example, the *nvdLookup* class can be instantiated from the command line allowing the direct lookup of vulnerabilities for a given software component rather than just through the MulVAL Code Generator software. The network modeller may leverage this capability to look up vulnerabilities on a specific machine by making a call to a *nvdLookup* object and report software vulnerabilities directly.

```
./nvdLookup [service name] [nvd directory]
```

4.5 Lessons learned

The following notes comprise lessons that have been learned or issues that remain to be addressed as part of the creation of the MulVAL Code Generator.

4.5.1 Software Name Agreement

A major challenge is to get agreement across the network XML, the NVD files and the resulting MulVAL expression for the naming of elements like the service (or software) name. Since the vulnerability database is external to the project, and therefore out of the control of the development team, it is easiest to follow the NVD convention for the naming of these elements. The NVD XML structure defines a 'product' tag in the following way

```
<vuln:product>
   cpe:a:microsoft:internet_information_server:6.0
</vuln:product>
```

If the NVD is to be the main source for vulnerability information, it would be beneficial to match this information where possible. The MulVAL Code Generator uses the service attribute to represent software information (e.g. microsoft internet information server) and the version attribute. When creating the tag to perform the vulnerability lookup, these two attributes are merged to create the software version that will be used to search the NVD data files.

4.5.2 Use of Capital Letters

XSB interprets values that begin with capital letters as variables. Some data elements, like CVE numbers, are capitalized. Therefore, when generating Datalog with these values, one of two approaches must be taken. Either the values must be translated to lowercase or they

must be encased inside quotes. The MulVAL Prolog Generator uses translated values since unnecessary use of quoted values may result in lowered XSB performance.

4.5.3 Vulnerability Aggregation

When pulling in vulnerabilities associated with a particular software component, there are two approaches: vulnerabilities can be manually added to the network model or vulnerabilities can be automatically added to the model. When using the automatic approach, the logic is to match the software to the NVD source and import ALL vulnerabilities that are listed for that product version (see Section 4.1.1: Design Principles). This method of pulling in all associated vulnerabilities was the chosen method for the MulVAL Code Generator. However, there are limitations to how the NVD represents software components. When assigning vulnerabilities to software, the NVD has 3 categories of components:

- OS level components,
- applications and
- hardware.

However, the demarcation point between these categories is not well defined. For example, Active Directory is considered to be an OS level software component. That would imply that to capture all software vulnerabilities associated with applications, we would need to import all operating system vulnerabilities in addition to all the application level vulnerabilities. The problem arise, then, that since the OS level category lists vulnerabilities for a very wide set of software components (e.g. Active Directory), then ALL systems in the network model would indicate the presence of all vulnerabilities associated with that operating system and not reflect the true situation. This problem exists not only vertically (from the application to the OS category) but also horizontally between applications. That is, some applications capture a wide set of sub-application components and it is not possible to separate the application from the components. For example, in NVD the Microsoft Application Server (MAS) is considered a component of Microsoft Internet Information Server. To capture Application Server vulnerabilities, we would have to state that all IIS installations have this vulnerability, even if the MAS component is not installed. Currently, the only method to detect what component a vulnerability is associated with is to scan the vulnerability text description. More investigation is recommended to address this issue. Currently, only two solutions are being considered: either move the model to a more manual vulnerability selection approach or scan the text description for applicable vulnerabilities. One other solution would be to attempt to match the vulnerability to the presence of certain protocols/ports on the system, but this increases the complexity of the solution and may lead to Type II errors (false negatives).

5 Model Building

This section describes the activities completed under the fourth work track, this is, the Model Building track. Specifically, the work performed under this track included testing to validate the MulVAL output process used the sample Network Models presented in the *Network Scenarios Architecture for Security Research Testing Report*. These models, presented in the correct network XML format, were sent through the MulVAL Code Generator, updated to only include those vulnerabilities that would trigger the documented attack scenario and run through the XSB-based MulVAL Datalog Engine.

During the process of this work track, the network XML format was updated to include MulVAL-specific elements. In this case, the only element needed is a data component to identify the location of the attacker. This element, which resides at the same level as the site elements, contains a single child element that defines the site and machine location of the attacker.

```
<mulval>
  <attacker site='site' location='machine' />
</mulval>
```

Note that the same NVD file storage directory was used for each model run. This directory held 4 years worth of NVD vulnerability files:

1. nvdcve-2.0-2006.xml
2. nvdcve-2.0-2007.xml
3. nvdcve-2.0-2008.xml
4. nvdcve-2.0-2009.xml

5.1 10 Host Model

The 10 host model (version 1.1) was used as the basis for the this model test. MulVAL Datalog code was generated through the following command:

```
generateProlog.py 10host.xml 10host.mulval nvd
```

The execution time was approximately 18 seconds and associated 254 separate vulnerabilities with the systems in the 10 host network model. With an attacker located on the VPN segment MulVAL finds the following attack paths:

Figure 3: *10 Host Network Model Attack Graph*

5.2 200 Host Model

The 200 host model (version 1.1) was used as the basis for the this model test. MulVAL Datalog code was generated through the following command:

```
generateProlog.py 200host.xml 200host.mulval nvd
```

The execution time was approximately 19 seconds and associated 423 separate vulnerabilities with the systems in the 200 host network model. The output file from this process was updated to remove all vulnerabilities except the vulnerability identified as CVE-2007-2897. This vulnerability is described (within NVD) in the following terms:

> Microsoft Internet Information Services (IIS) 6.0 allows remote attackers to cause a denial of service (server instability or device hang), and possibly obtain sensitive information (device communication traffic); and might allow attackers with physical access to execute arbitrary code after connecting a data stream to a device COM port; via requests for a URI containing a '/' immediately before and after the name of a DOS device, as demonstrated by the /AUX/.aspx URI, which bypasses a blacklist for DOS device requests.

Since this vulnerability has a PARTIAL confidentially impact (again as defined by NVD), it is registered by our rules as a privilege escalation attack. This vulnerability is deemed to reside on the admin6 server, as defined by the attack scenario from the Network Modelling Report. With the updated vulnerabilities, which reflect the attack scenario, the run through the MulVAL Prolog Engine finds the following attack path:

```
<1>||--execCode(medium_admin_6,root)
  (2) RULE 0 : When a principal is compromised any machine he has an account on will also be compromised
    <3>|--canAccessHost(medium_admin_6)
      (4) RULE 7 : Access a host through executing code on the machine
        execCode(medium_admin_6,root)==><1>
    [8]-hasAccount(sysAdmin,medium_admin_6,root)
    <6>|--principalCompromised(sysAdmin)
      (7) RULE 10 : password sniffing
        execCode(medium_admin_6,root)==><1>
        [8]-hasAccount(sysAdmin,medium_admin_6,root)
  (9) RULE 2 : remote exploit of a server program
    <10>||--netAccess(medium_admin_6,https,httpsPort)
      (11) RULE 5 : multi-hop access
        [12]-hacl(medium_admin_6,medium_admin_6,https,httpsPort)
        execCode(medium_admin_6,root)==><1>
      (13) RULE 6 : direct2 network access
        [14]-hacl(medium_wap,medium_admin_6,https,httpsPort)
        [15]-attackerLocated(medium_wap)
    [16]-networkServiceInfo(medium_admin_6,internet_information_server_6_0,https,httpsPort,root)
    [17]-vulExists(medium_admin_6,cve_2007_2897,internet_information_server_6_0,remoteExploit,privEscalation)
```

From this result, it is clear that the MulVAL Generator can successfully translate from XML to MulVAL-ready Datalog and achieve valid results.

5.3 200,000 Host Model

The 200,000 host model (version 1.0) was used as the basis for this model test. MulVAL Datalog code was successfully generated through the following command:

```
generateProlog.py 200000host.xml 200000host.mulval nvd
```

The execution time was approximately 185 seconds and associated 607 separate vulnerabilities with the systems in the 200,000 host network model. The resulting Datalog file consisted of 2,413,764 separate Datalog statements. When all vulnerability statements were removed (vulExists, vulProperty), the resulting file contained 226,061 statements. When attempting to load this file into XSB/Mulval, the XSB system crashed with the following error:

```
Running attack simulation...
pointsto wrong space error during unchaining
Exiting XSB abnormally...
```

This error is consistent with behaviour seen when XSB has run out of memory, implying that the 200,000 model (including the use of machine templates) without vulnerability information is too large to be handled by XSB. It is recommended that either a smaller ”large network model” be used or research into alternate XSB solutions or configurations be made to find a large model that can withstand logic engine analysis.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Bell Canada
160 Elgin St., 17th Floor, Ottawa, Ontario, K2P 2C4

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

UNCLASSIFIED
(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC June 2010

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Security Posture Assessment Demonstrator and Experimenter

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

Bacic, E.; Henderson, G.; Tremblay, L.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
May 2011	34	0

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Contract Report

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Ottawa
3701 Carling Avenue, Ottawa ON K1A 0Z4, Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
15BB03	W7714-071028

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Ottawa CR 2011-011	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

There is a recognized need within the network defence community for analysis tools that can assist in the evaluation of networks for vulnerabilities and aid in the design of robust networks. This paper describes the initial design and development work to create a unified framework that can leverage research, commercial and academic security analysis tools in support of ongoing network defence analysis practices. An initial examination of network modeling and model manipulation techniques is provided, including: appropriate data representation, automated and interactive interfaces and necessary modeling capabilities. The analysis portion of the proposed framework is described both in terms of its overall pluggable architecture and in terms of sample modules that could be integrated within and leveraged by the overall analytical framework. The data storage mechanisms for model data is presented, not only for data generated through the analysis process, but also the means by which external data sources can be leveraged to provide near real time contextual data for the various analytical modules. To illustrate the framework in action, data flow and transformation for the use of a specific module, the MulVAL analysis engine, is described. A series of network models were tested against the framework and sample module showing the validity of the design approach.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

JNDMS; AssetRank; MulVAL; RapidMiner; network models

Defence R&D Canada

Canada's leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca