YANG-Based Configuration Modeling - The SecSIP IPS Case Study

Abdelkader Lahmadi, Emmanuel Nataf, Olivier Festor

YANG-based Configuration Modeling
- The SecSIP IPS Case Study -

Abdelkader Lahmadi, Emmanuel Nataf, Olivier Festor
The MADYNES Team
LORIA - INRIA Nancy Grand Est
615 rue du jardin botanique, 54602 Villers les Nancy
France
Surname.Name@loria.fr

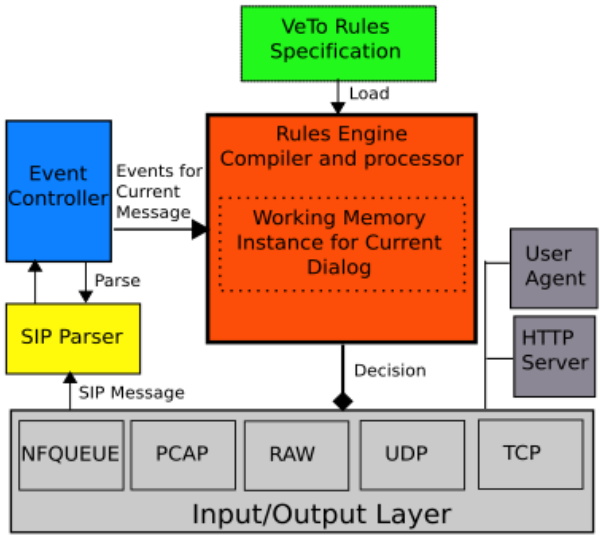
IM 2011, 12th IFIP/IEEE International Symposium on Integrated Network Management

Abstract

We present our experience with the development of an XML-based configuration model for an Intrusion Prevention System (IPS) dedicated to the Session Initiation Protocol (SIP) used in voice over IP signaling. In previous works [AL-IM09, AL-NOMS10] we have presented the SecSIP framework, a prevention system for SIP-based networks, which adopts a rule-based approach for specifying preventions on SIP protocol activities to stop attacks exploiting known vulnerability before reaching their targets. The SecSIP framework relies on a proprietary language called VeTo to express the prevention rules. SecSIP uses a plain text configuration file in which specifications are authored and managed manually. While extending the deployment of the framework beyond our own lab, support for remote configuration was required. Given the promise of Netconf, we naturally turned our investigations towards this protocol and embraced the YANG data-modeling framework. In this paper we present the modeling result on the SecSIP configuration interface and share our experience with both YANG and Netconf.
The first part of the paper is dedicated to the description of the data to be modeled, namely VeTo policies. The second part presents the Yang model built for VeTo policies and the Netconf framework put in place. Lessons learned during both modeling and coding phases are presented in a third part of the presentation. Finally some conclusions are given and future work is outlined.

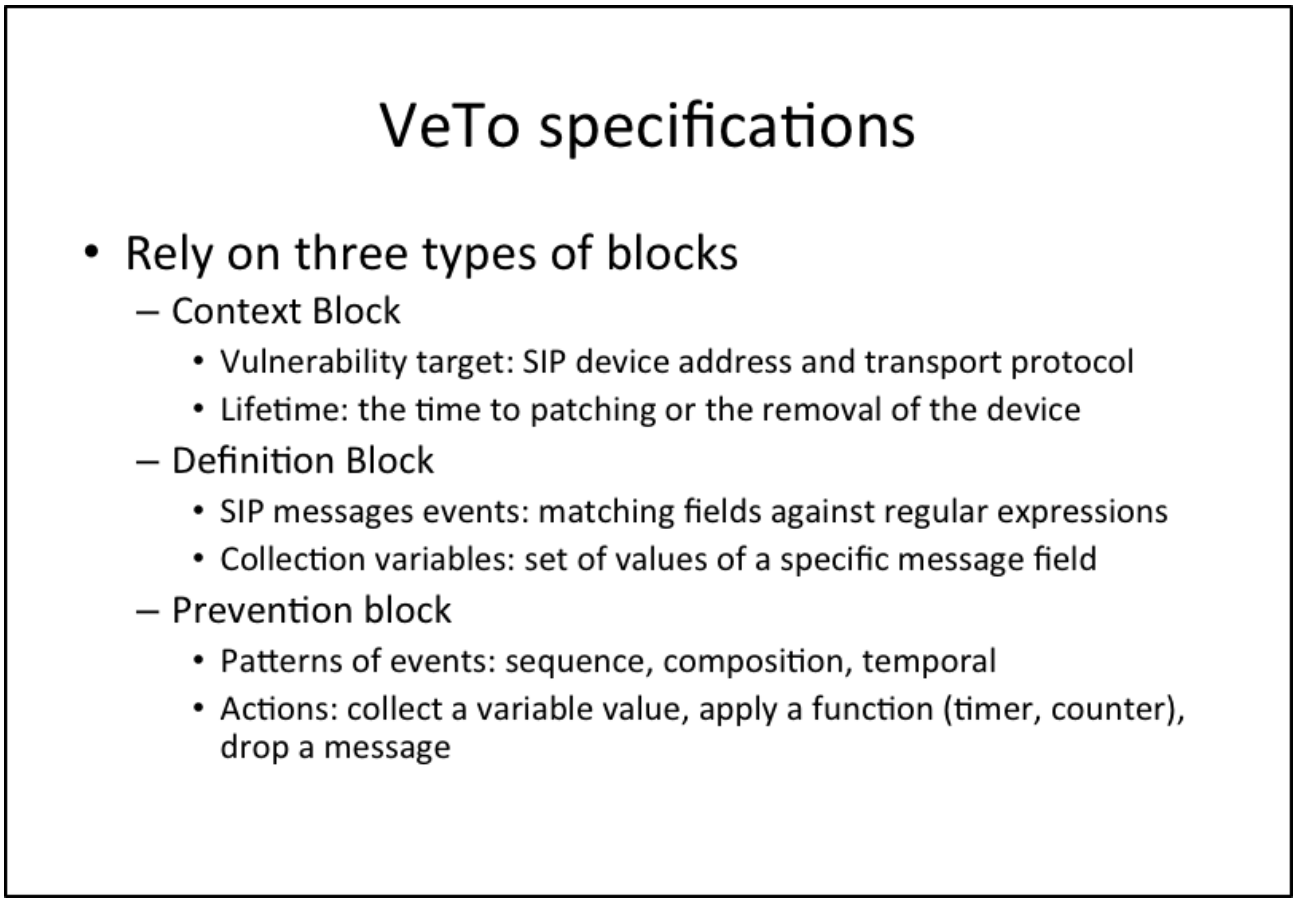
Context and Motivations

- SecSIP is a protection environment dedicated to SIP
  - Uses a proprietary language to describe protection specifications
- Large set of policies specification
  - error prone
  - Ad-hoc monitoring agent



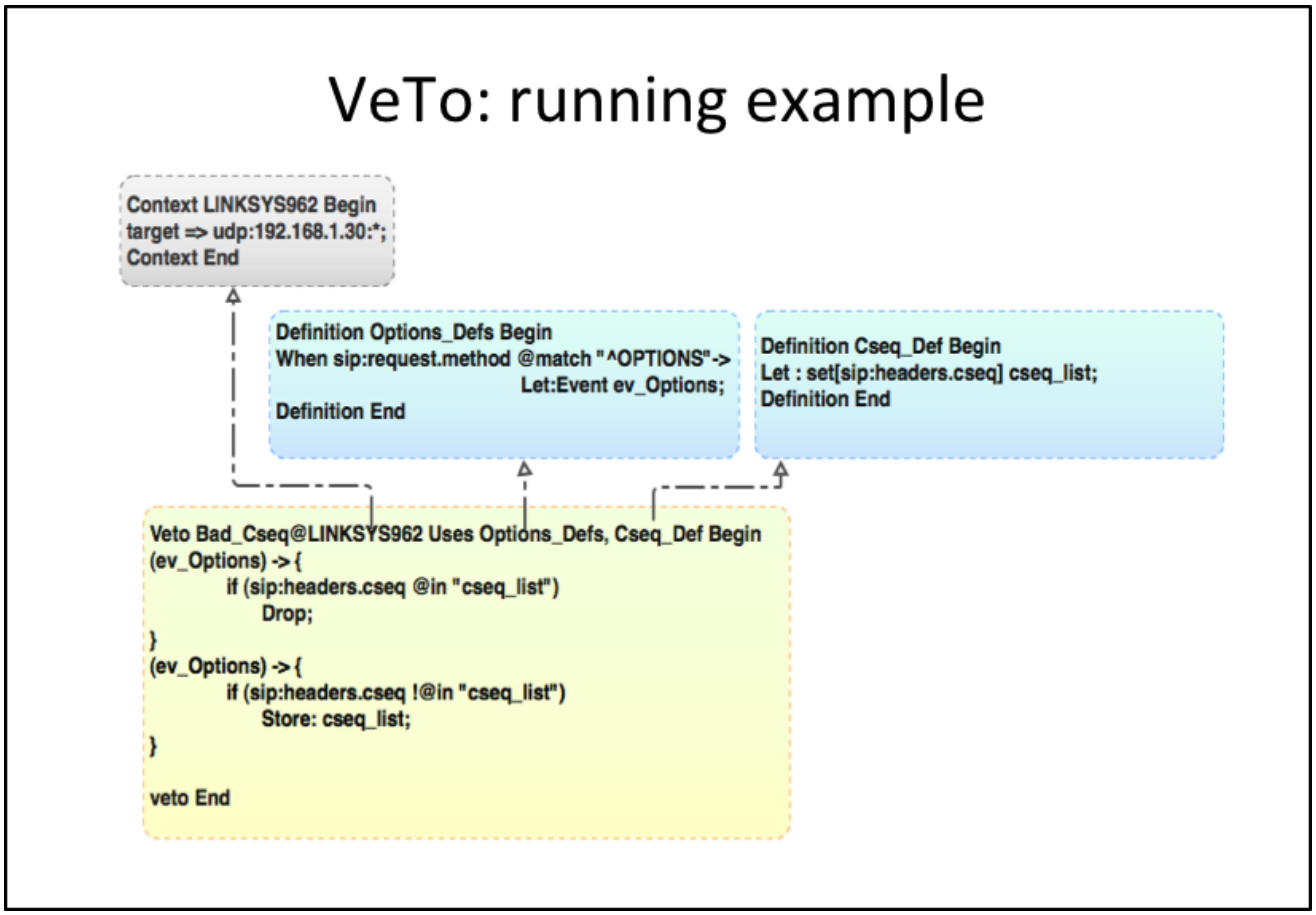
We need a standard data modelling format to manage SecSIP configurations and internal states

The objective of the SecSIP framework is to defend SIP-based services against exploitation of known vulnerabilities. SecSIP provides a proactive point of defence between a SIP-based network of devices (servers, proxies, user agents) and the open Internet. There, the entire SIP traffic between the enterprise network and the outside world is inspected and analyzed against authored VeTo specifications before it is forwarded. When initializing, the SecSIP runtime starts loading and parsing authored VeTo blocks to identify different variables, event patterns, operations and actions from each rule. It implements an input and output layer, to capture, inject, send and receive SIP packets from and to the network. Intercepted packets are moved to the SIP Packet parser module. The main function of this module is to extract different fields within a SIP message and trigger events specified in the definition blocks. During each execution started by the arrival of a SIP message, the SecSIP runtime uses a data flow acyclic graph to identify input matching rules and triggers their respective events. The paired events in each operator node are propagated over the graph until a pattern is satisfied. When the pattern is satisfied, the respective rule is fired and the set of actions is executed. SecSIP offers a CLI access to its configuration. To ease the remote configuration of SecSIP boxes, we designed both a YANG model and the corresponding support in a Netconf agent.



Let's start with a brief overview of the VeTo language syntax and semantics. The language relies on an event-driven and rule-based approach to specify in a flexible and a scalable manner, prevention rules from existing vulnerabilities within a SIP network. The language combines context, definitions and events blocks extracted from the properties of vulerabilities to provide the ability to prevent against its exploitation. The context block exhibits the vulnerability surrounding environment properties. The definition block provides the vulnerability related assumptions on its behaviour such as the involved SIP messages and their respective fields.

The prevention block describes the vulnerable behaviour within its context and includes a response action. We have shown through real vulnerabilities and exploit code deployed the usage and efficiency of VeTo specifications to protect different deployed SIP devices on a target testbed. These blocks are linked since the protection block relies on the definition and context blocks to describe a protection. Each vulnerability is specified using a single prevention block containing event patterns based rules to describe a vulnerable behaviour of SIP activity. Each rule header contains the event pattern and its body contains a set of actions to be applied when the pattern is satisfied. These rules may also reference collection variables that contain a specific message field values collected from different observed SIP messages. The variables values are kept in memory during an active SIP dialog. The different variables and events used by a prevention block are defined in one or many definition blocks. Using these different block types we are able to specify a vulnerability exploitation behaviour and its countermeasure when it is detected in a SIP activity.



In this tiny slide, we present an example of a VeTo specification to prevent the exploit of a vulnerability target a SIP-based hardphone. The vulnerability exploitation here consists in sending to the SIP phone multiple SIP messages of type *OPTIONS* with the same value of the *cseq* field. This leads to a crash of the phone.

As mentioned in the previous slide, we describe a vulnerability using at least three blocks. The first block is the context where we specify the target identifiers, mainly the transport protocol used to carry SIP messages to the device and the IP address listening port tuple. Herein, we used two definition blocks. The first block defines an event *ev_Options* when a SIP message is of type *OPTIONS*. The second block defines a collection of values of types set which contains the values of the SIP field *cseq*. Finally, we define a prevention block called *Bad_Cseq*. This block is attached to the context block *LINKSYS962* and uses the two definition blocks *Options_Defs* and *Cseq_Def*. The block defines two rules. The first rule drops the message when an *ev_options* event is observed for which the collection list contains already the current value of the *cseq* field. The second rule stores the current value of the *cseq* field of a SIP message of type *OPTIONS*. When the SecSIP engine loads these blocks, it will start to parse and execute them against the observed SIP network towards the networked devices.

SecSIP internal data states

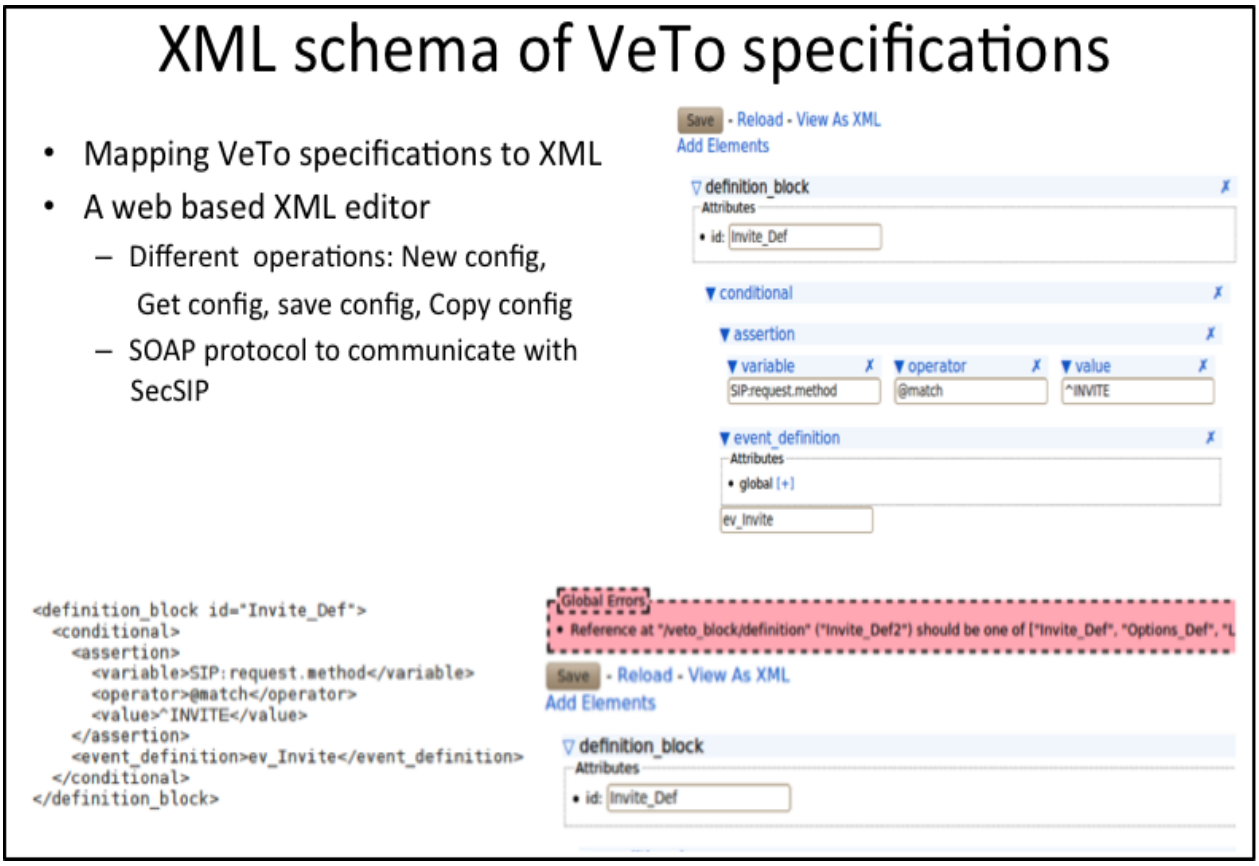
- Different variables associated to each active dialog
  - Collection, counter, timer, event

- Event graph
  - Event patterns matching
  - Graph instance per dialog
  - Root nodes: events
  - Intermediate nodes: operators
  - Leaf nodes: actions

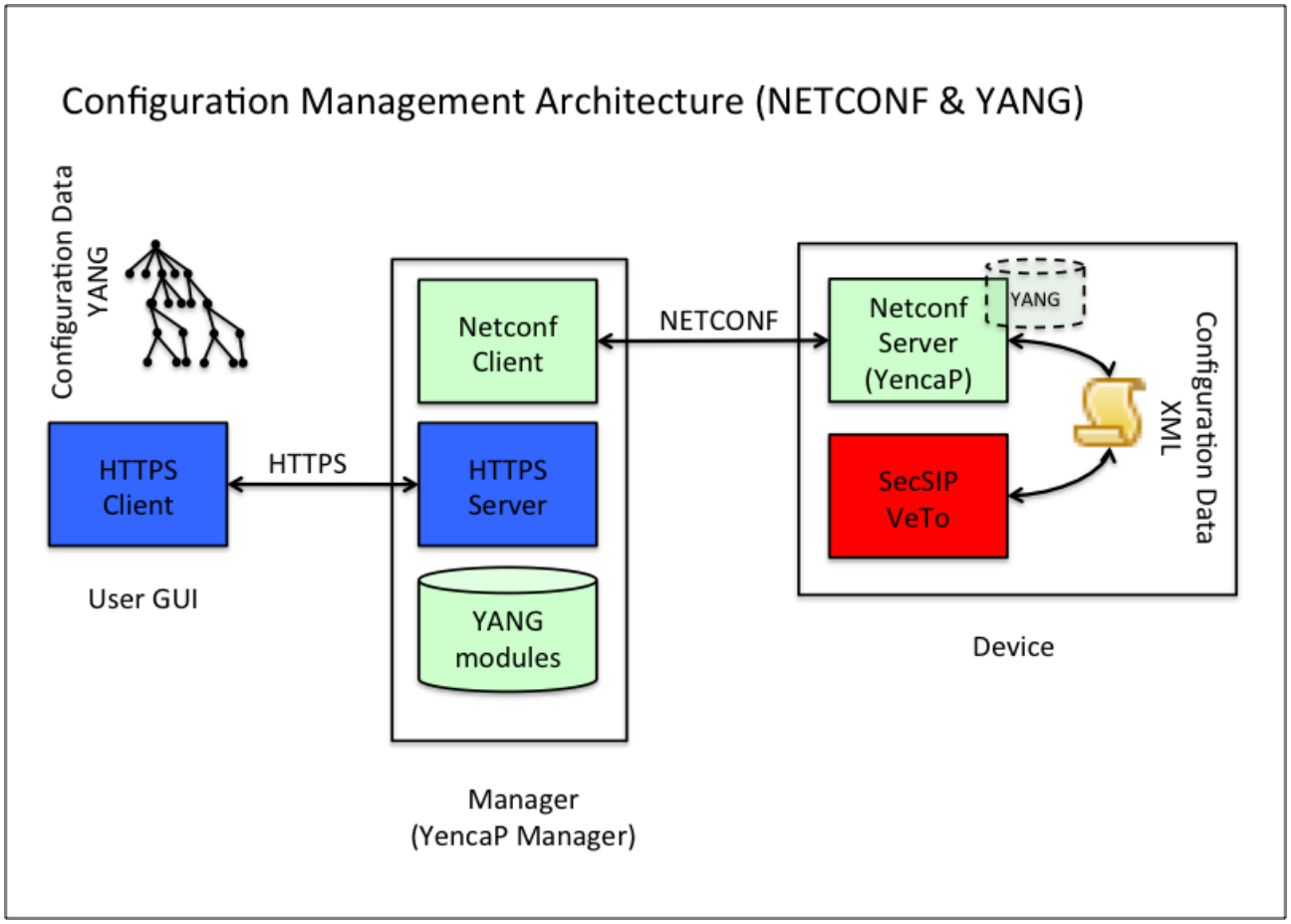
In the previous slide, we showed how to author a set of VeTo rules to specify a vulnerability prevention. When these rules are loaded by the SecSIP runtime, it starts executing those rules against the observed SIP traffic. These rules may contain different variables to be instantiated for each active SIP dialog. The most interesting feature in the SecSIP engine is to efficiently compute event patterns against SIP messages. We use a graph based matching approach to match the defined patterns within the different authored prevention blocks. SecSIP uses a global graph to track observed events.

The graph is built sequentially. Each time a rule is parsed and a node has to be inserted, SecSIP first checks if a similar node exists in the graph. If such node exists with the same parameters and the same parents it is simply reused.

This graph is a merge of the different event patterns defined by the different VeTo blocks. Each root node in the graph represents an event. Intermediate nodes represent the composition of these events as described by the event patterns. Finally, the leaf nodes are the actions to be done when a specific pattern is satisfied. Links between nodes are labeled to denote the position of the event within a pattern (a left or right first event). When an event is observed, a token is placed in its nodes. Then, this token starts to travel within the graph until it reaches a leaf node where an action has to be executed.



To author VeTo specifications, we developed a web-based manager. The manager is a Java script web application and it is served by SecSIP using a nanohttp library. It is built using the JavaScriptMVC framework, which provides an infrastructure that uses the model-view-controller design pattern. It relies on a XML Schema to express and verify the grammar used to create and edit configurations. The manager communicates with SecSIP using SOAP. We have build different models to represent the information that the manager works with. The Config model handles communications with SecSIP, and deals with configurations as a whole. It has methods to get and send a configuration to SecSIP, and copy a configuration onto another one. The Rule Element model represents a part of a configuration. In the manager, the representation of a configuration is similar to an XML tree: it has a Rule Element at its root, and this Rule Element can contain attributes, a text value, and other Rule Elements. Finally, the Constraint model represents all constraints that have been expressed in the XML Schema. It checks that mandatory children and attributes of Rule Elements are there and in the correct order, that no alien elements are present, and it can also test reference and uniqueness constraints that can be present in XML Schemas. To render those models to the user, a couple of views have been created, and for each view, a controller has been added, that interacts with the models. The Config view and controller consist of the menu bar at the top of the manager window, and link each button in the menu to one of the methods defined in the Config model (new config, get, send, copy config). The Root Element view and controllers are used to display configuration inconsistencies and errors detected by constraint that are defined at the root of the XML Schema, or that span over multiple VeTo blocks. The Tree view and controller are used to render and interact with the list of VeTo blocks that appears at the left of the manager window. The main function of this list is to allow the user to select individual blocks to view and edit them, and it also allows creating new blocks, and to delete and reorder existing blocks. Finally, the Element view and controller are used to display and interact with Rule Elements, which represent the actual content of the configuration.



The architecture of our standardized configuration management is made of three distributed parts: the User GUI, the Manager and the Device. We use the IETF standardized approach to configuration management with the NETCONF [RFC4741] protocol and its data modeling language called YANG [YANG].
The graphical user interface allows to establish an HTTPS connection with the Manager in order to handle configuration data that are modeled with YANG [Nat-NOMS10]. This interface shows the YANG data tree that reflects current configuration data values of the Device.
In the Manager, the HTTPS server acts as a proxy to access devices with the NETCONF protocol and save to a database of YANG data models (called modules).
The NETCONF server is embedded with the Device and its configuration data is XML formatted. The VeTo engine uses these data as definitions and protection rules. Configuration data are modeled by a YANG module that must be known by the YANG modules database of the Manager in order to match XML configuration data conveyed by NETCONF with YANG data models.
The open source applications YencaP and YencaP Manager implement both NETCONF server and client side [YENCAP]. The Manager has to connect to a NETCONF server on the behalf of the user and has to retrieve which YANG modules are implemented in the Device. The Manager is then enable to look for these modules in its database and to build a java applet that will be returned to the user GUI. After that, all requests from the User GUI are conveyed in a HTTPS request that contain NETCONF operation and the Manager has to replace HTTPS headers by NETCONF headers and to send the request to the NETCONF server. The response is processed in the opposite way.

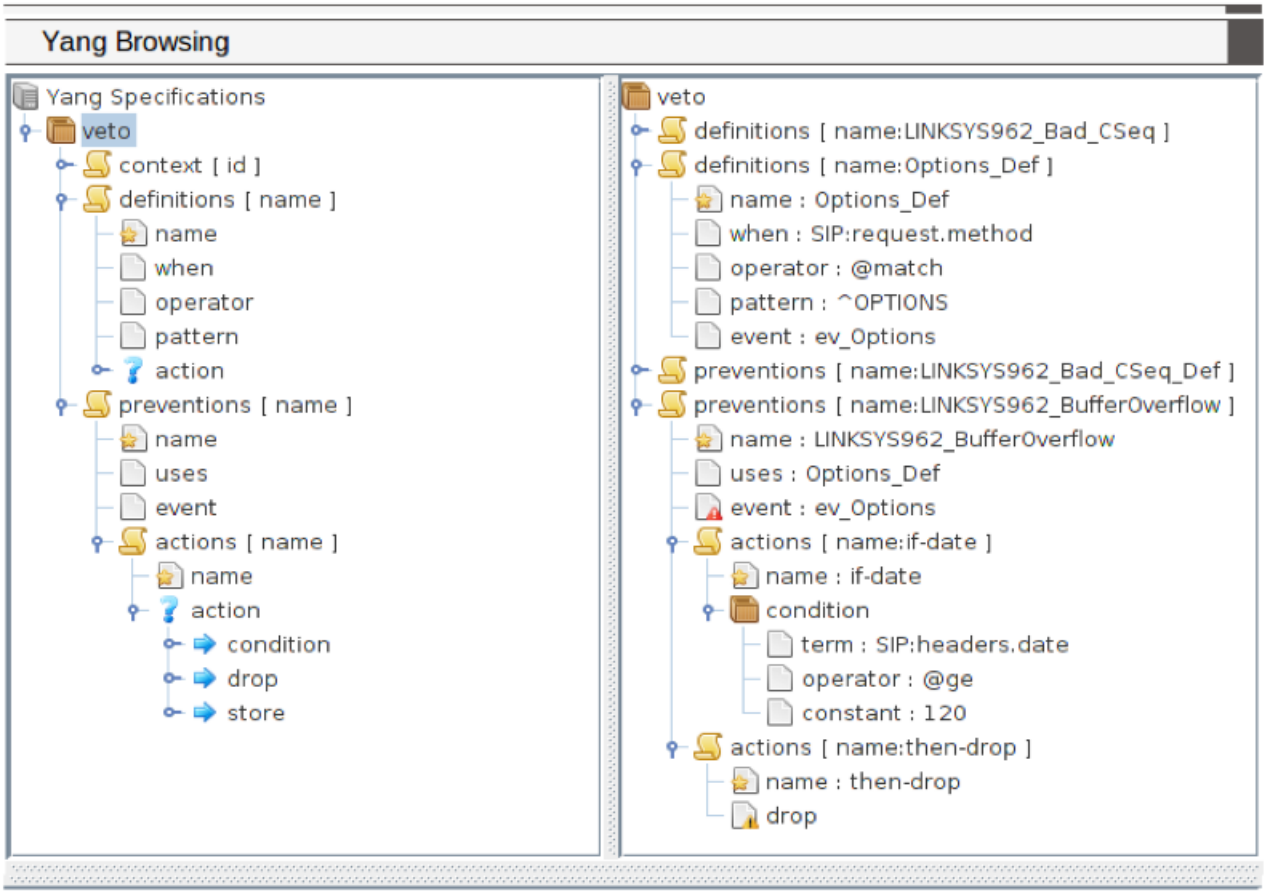
YANG model of VeTo rules

```
1  typedef vtTerm {
2     type string;  {
3        pattern   "SIP:(request|response|message)\. "
   +
4                  "(method|status-code|uri|Call-ID)" ;
5     }
6  }
7  typedef vtOperator {
8     type string {
9        pattern « @(ge|le|in|match) »;
10    }
11 }
12 list definitions {
13    key name;
14    leaf name { type string; }
15    leaf when { type vtTerm; }
16    leaf operator { type vtOperator;}
17    leaf pattern { type string; }
18    choice action {
19       leaf event {type string;}
20       case collection {
21          leaf collectionName { type string;}
22          leaf collectionType { type string;}
23       }
24    }
25 }
26 list preventions {
27    key name {
28    leaf name {type string;}
29    leaf-list uses { type leafref {
30       path «../../definitions/name »;}
31    }
32    leaf event { type leafref {
33       path « ../../definitions[name = current()/../uses]/
   actions/event  »;}
34    }
35    list actions {
36       choice action {
37          container condition {
38             leaf term { type vtTerm;}
39             leaf operator { type vtOperator;}
40             choice operand {
41                leaf constant { type string;}
42                leaf variable {type leafref {
43                path «../definitions[name = current()/../uses] »
44                      + « /collectionName »; }
45             }
46          }
47          leaf drop { type empty;}
48          leaf store{ type leafref {
49             path  «../definitions[name = current()/../uses] »
50                   + « /collectionName »; }
51          }
52       }
53 }
```

VeTo rules are modeled within one YANG module. This listing shows a subset of the full YANG module. YANG allows the definition of type with the typedef statements (lines 1-11) and we use this to model some parameters of VeTo. For example the vtTerm definition models the selection of SIP fields with a string following the regular expression given in the pattern statement (lines 3 and 4). The next statement models any VeTo definition block (lines 12-25) with a list where each line contains the values of the `when` operator and pattern used to trigger one action. As action could be of several natures like an event or the creation of a collection, we model the action by a choice statement with one case by possible action. Here are the action of sending an event (line 19) and the action of creating a collection (line 20-22) with a name and a type.

The second list (from the line 26 to the end) models any VeTo prevention rules. The part we show here focuses on the relationship between preventions rules and definitions. The leaf-list statement (lines 29-31) models the use of definitions by the rule. A leaf-list is a list of several value of the same type and the leafref type is for referencing data in the YANG data tree. The path statement (line 30) gives a relative path to all definition names and so ensures that a protection rule uses only existing definitions. It is the same for the leaf event (lines 32-34) because we need to ensure that the event of this rule is defined in one of the used definitions. The path statement (line 33) specifies that there is one definition whose name is in the leaf-list uses. It is worth noting the relative notation in the Xpath expression and the use of predicate.

Finally we model protection actions with a list of actions (lines 36-53) that can be conditional, where we specify that a condition can be expressed on constant or already defined collection (lines 42-45). The action can simply be a rejection (line 48) or to store in an existing collection (same Xpath expression lines 50-51).



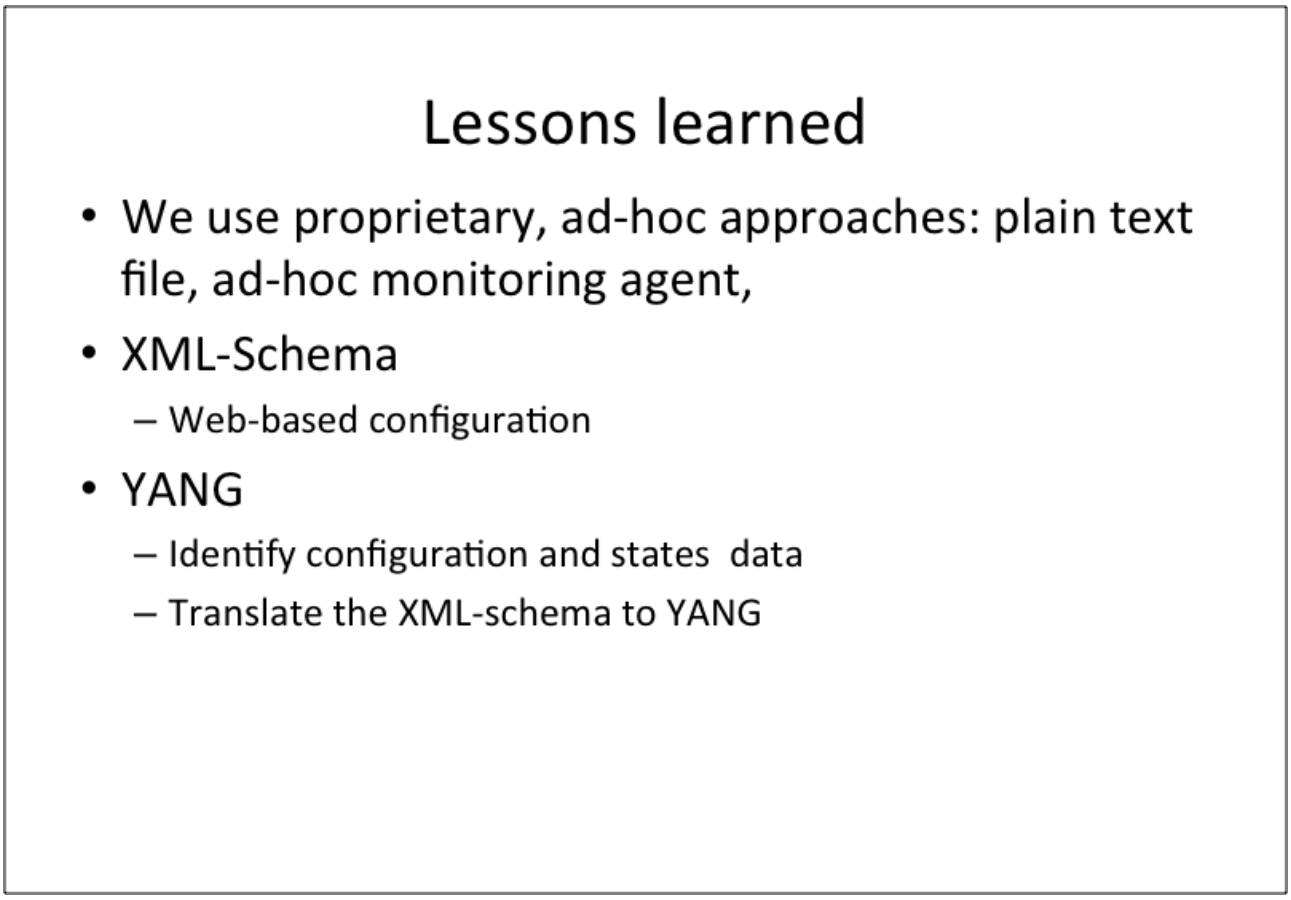
The YANG graphical user interface is implemented as a java applet running on the user manager station (User GUI in previous figure). The left part is the YANG data model and the right part is the YANG data tree of selected parts in the model.

The data model visualization is a tree model that trivially fits itself to the schema tree of YANG models. Any YANG modules have the same virtual root called “Yang Specification” and modules can be browsed with simple mouse action. Icon faces are related to YANG statements, as a box for a container or a “?” for a choice.

The right part is also a tree but of configuration data values. One can note that only data values (called YANG data tree) are displayed and there is no choice and case statements in this view. The view shows two instances of each list “definitions” and “protections”.

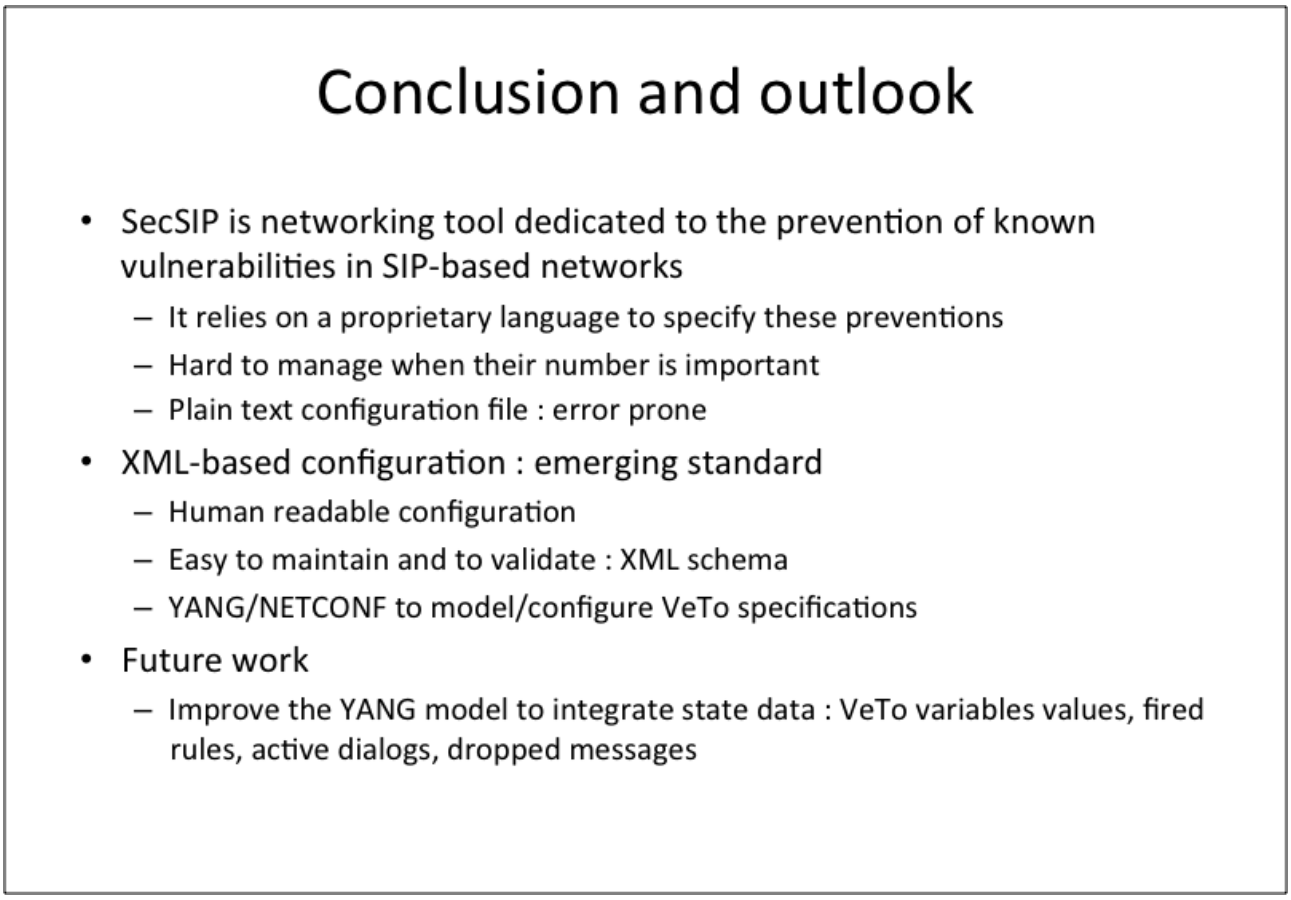
This same view is used to create new value, as for example a new line or to update existing values. Syntax checking could be made before sending the request. The applet can also check some semantic constraints like the Xpath expressions in the path YANG statements but only on data currently displayed. This version is more a light management client and cannot locally manage a whole configuration. Such global tree constraints checking should be done at the server side and this is an ongoing work.

As configuration data could be huge, we also plan to replace this simple tree based view by more sophisticated graph representations that are available for web-based interfaces.



Although, in this work, SecSIP serves as a concrete case study for an IPS tool dedicated to SIP protocol, it is like a majority of networking tools (Snort, Bro, Prelude) which rely on proprietary languages to specify the preventions against attack exploits. These specifications are usually authored in their respective language in a plain text file updated by the user or an administrator. Maintaining these specifications becomes quickly tedious and error prone when the number of specifications becomes important. Therefore, in this work, we have shown the use of standard configuration data models like XML schema and YANG to represent these prevention specifications. The success of this approach depends on the complexity of the specification language to be translated into an XML-based modeling language. Using such language allows us to support web based management tools.

In our approach, we used the following steps: first, we translated VeTo specifications to an XML-schema. Secondly, from this XML schema we deduced a YANG data model that can be used by the NETCONF protocol. When developing the YANG model, we found that we could directly model VeTo specifications to a YANG data model without going through an XML schema step. Therefore, YANG seems to be a good candidate to model complex configuration data like VeTo specifications.



In this work, we have described a configuration management data model for the SecSIP tool that we have developed to prevent the exploit of existing vulnerabilities of SIP protocol implementations. This tool uses a language called VeTo to specify these preventions. We found that authoring these specifications using a plain text file could be suitable when the number of vulnerabilities is small. However, dealing with an important number of prevention specifications, a file-based configuration becomes error prone and hard to maintain. Therefore, we have constructed an XML based model of these specifications. Then, this model has been easily translated to the YANG data modeling language. We have used the resulting YANG model in the YencaP framework to configure SecSIP.

Currently, our model only focuses on the configuration data. In a future work, we have to improve our YANG model to integrate the state data to expose SecSIP internal states like: VeTo variables values, fired rules and active dialogs.

References

[AL-IM09] A. Lahmadi; O. Festor, SecSip: A Stateful Firewall for SIP-based Networks, 11th IFIP/ IEEE International Symposium on Integrated Network Management, IM'09, Hofstra University, Long Island, NY, USA, June 1-5, 2009. IEEE 2009

[AL-NOMS10] A. Lahmadi, O. Festor, VeTo: An exploit prevention language from known vulnerabilities in SIP services, Network Operations and Management Symposium, NOMS'2010, April 19-23, 2010, Osaka, Japan

[RFC4741] R. Enns, NETCONF Configuration Protocol, RFC 4741, December 2006
[YANG] M. Bjorklund,YANG - A data modeling language for NETCONF,
draft-ietf-netmod-yang-13, Network Working Group, Internet-Draft, June 1, 2010

[YENCAP] V. Cridlig; R. State, YencaP Documentation
Technical Report, 2005, 25 Pages,http://hal.inria.fr/inria-00000804/fr

[Nat-NOMS10] E. Nataf; O. Festor, End to end Yang-based Configuration Management, Network Operations and Management Symposium, NOMS'2010, April 19-23, 2010, Osaka, Japan