Symbolic Reasoning in the Cyber Security Domain

June 2007

Michael Kandefer, Stuart C. Shapiro, Adam Stotz, and Moises Sudit
{mwk3,shapiro}@buffalo.edu, stotz@cubrc.org, and sudit@eng.buffalo.edu

Department of Computer Science and Engineering, Center for Cognitive Science, and
The National Center for Multisource Information Fusion.

University at Buffalo, Buffalo, NY, 14260-2000.

ABSTRACT

Cyber Security can benefit greatly from the association and combination of data and information from multiple sources. A data repository of system vulnerabilities, a network scanning tool, and the advice of a systems analyst trained in cyber security can all aid in identifying and preventing intruders. Previous attempts at information fusion in cyber security have largely concerned themselves with the "tangible" information sources, but this ignores an important resource in solving problems in this particular domain --- the cyber security expert's reasoning process. The National Center for Information Fusion (NCMIF) has begun implementing a solution that partially automates the cyber security expert in the intrusion detection process through a combination of information fusion techniques and symbolic reasoning, using the SNePS knowledge representation, reasoning, and acting system.

Our methodology approaches cyber security problems by fusing information from external information repositories into a SNePS-based agent's knowledge base. We have identified five information sources that are useful: the background knowledge of a cyber security subject matter expert (SME); Nessus security scan reports; the Common Vulnerabilities and Exposures (CVE) database; and INFERD template graphs. The SNePS system makes use of higher-order logic to represent information about the external world. Facts are represented as proposition-valued terms, and the SME's reasoning procedures are represented as logical rules.

We have also developed a Graphical User Interface to SNePS that facilitates the development and maintenance of the knowledge base. Dissemination of knowledge bases will be accomplished via files using OWL and the RDF markup language.

1.0 Introduction

The National Center for Multi-source Information Fusion (NCMIF) [1] has endeavored to combine the techniques of information fusion and knowledge representation and reasoning to provide a solution to problems in the cyber security domain. We present here a subtask in this goal. This task is responsible for providing a cyber security knowledge base that can test INFERD [7] template graphs in order to gain a better understanding of a hacker attack. For our purposes we have selected the SNePS Knowledge Representation and Reasoning System [6] to accomplish this task, which can be further subdivided as follows:

- Represent the selected information in SNePS.
- Use the SNePS reasoning engine to aid information fusion and cyber security.
- Provide a method of interfacing with the SNePS system.
- Improve the system, as needed, to accomplish the previous.

Fusing the information sources into a common, logical representation language allows our reasoning engine to treat disparate sources as if they were one, and reason about the contents of these sources. In addition, we have developed two methods of interfacing with the system, a Java-to-SNePS API and a SNePS GUI. The former allows for Java programs to assert information in the SNePS system and construct logic-based queries to the system. The SNePS GUI provides users various alternative methods of inserting information into SNePS using visualized SNePS networks and hierarchies. This paper discusses the information sources we are using, the representation of those sources, and the developed and forthcoming interface features.

2.0 Symbolic Reasoning in SNePS and Information Fusion

Representing information in symbolic logic not only allows for the contents of these information sources to be fused into a common representation language, but also allows various reasoning tasks to be performed on the logical representation. Logical rules can be applied to deduce new information from existing information. For example, given rules about the structure of a class membership hierarchy, a traditional logic-based system can determine class membership. SNePS is one such system, but also provides a number of other useful representational and functional facilities that can aid information fusion and reasoning in the cyber domain. These include the capabilities to:

- represent and distinguish co-referential terms;
- represent meta-knowledge;
- detect contradictions.

The SNePS system was designed to handle co-referential terms in a way that provides each term its own unique intentional denotation, but allows for these terms to co-refer using equivalence relationships. Such a representation technique can provide a *sense* for an entity that is unique to each information source. SNePS can also represent knowledge about knowledge. Such a feature is capable of representing what SNePS knows about the knowledge contents of the external information sources.

Apart from representational features, one functional feature of the SNePS system is its

capability for belief revision. The system can automatically detect contradictory information and present it to the user of the system. After the user selects the information to reject, an automatic repair propagation procedure is invoked. Such a feature is desired in information fusion as different external sources may contain contradictory information.

3.0 Representing the Information sources

We have identified five information sources that will aid in reaching our goals: the background knowledge of a cyber security subject matter expert (SME); Nessus security scan reports; Common Vulnerabilities and Exposures (CVE) database; SNORT Sensor rules; INFERD template graphs. A description of these and how they are represented follows:

3.1 SME Background knowledge

Of the five information sources the SME's background knowledge is the most difficult to elicit. In order to accomplish this we're taking our task one step at a time, analyzing the questions we want answered at the current step, and then asking the SME how they'd go about it. Through this process we have identified a number of external information tools to aid us, as well as constructed reasoning axioms that can answer the current questions under consideration. Presently, the process under consideration is determining if an INFERD attack track for a particular host is a false positive, or true-positive. We have determined with the help of the SME this can be done by examining the vulnerabilities of the system using the Nessus security scanning tool (Section 2.2) and comparing their CVE identifiers (Section 2.3) against the Signature identifiers (SID) provided in the attack track. In order to do this two translations are accomplished; one from SID to BID (Bugtraq ID), and one from BID to CVE identifier. This extra complexity step was chosen as the CVE repository has more entries with BIDs, than SIDs.

Simply put, if the CVE identifier provided by Nessus for a particular host does not match the SID identifier provided by INFERD for that host, the track can be regarded as a false-positive, otherwise it's a true-positive or that a vulnerability exists on the system but was not found by Nessus (a possibility we're discounting provisionally). This procedure was chosen since it is possible for INFERD to generate attack tracks for vulnerabilities the host in question doesn't possess, and as such, they should be rejected. The representation of this reasoning process uses proposition-valued terms with the following semantics:

- *PropertyValue(x,y,z)* - Entity *x* has a property *y* with value *z*
- *CVE_BID_Equiv(c,b)* - CVE identifier *c* refers to the same vulnerability that BID *b* refers to.
- *SID_BID_Equiv(s,b)* - SID *s* refers to the same vulnerability that BID *b* refers to.
- *TruePositive(x,c,s)* - Entity *x* has a true-positive when it is known to have a vulnerability referred to by a CVE identifier *c* and SID *s*.

This reasoning rule is expressed in SNePSLOG[1] as:

```
all(cid,bid,sid)(SID_BID_Equiv(sid,bid)
   => (CVE_BID_Equiv(cid,bid)
      => all(x)({PropertyValue(x,CVE,cid), PropertyValue(x, BID, bid),
                 PropertyValue(x, SID, sid)}
         &=>   TruePositive(x,cid,sid)))).

all(x,cid)(PropertyValue(x,CVE,cid)
  => all(bid)(CVE_BID_Equiv(cid,bid)  => PropertyValue(x,BID,bid))).
```

The above rules represent the following propositions: If a particular host *x* is known to have a particular SID *sid* , BID *bid*, and CVE identifier *cid*; and that *sid*, *bid*, and *cid* refer to the same vulnerability, then INFERD generated an attack track that is a true-positive. It is assumed by the INFERD interface to SNePS that after providing information about a particular attack track that if the system doesn't reason to a true-positive that the track is a false-positive.

Apart from the above reasoning rules, some general rules are provided to the system for the task at hand as well as in anticipation for future reasoning tasks. Among these are general "part of" and "class membership" rules. The two "part of" rules are as follows:

```
1) all(x,y,z)({PartOf(x,y),PartOf(y,z)} &=> {PartOf(x,z)}).
2) all(p,v,x,y)({SubsumedProperty(p), PropertyValue(x,p,v), PartOf(x,y)}
     &=>   {PropertyValue(y,p,v)}).
```

The first rule provides the system with the capability to reason that the "part of" (expressed as *PartOf* in our logic) relationship is transitive. For example, if host *h1* is part of network *n1*, and port *p1* is part of *h1*, it can be concluded *p1* is part of the network *n1*. The second rule is specifically tailored for properties the system believes are subsumed by parent parts (expressed as *SumbsumedProperty*). This rule was selected as Nessus reports only that ports have a specific vulnerability on them, but our desired goal is to know if certain hosts have vulnerability. By asserting that CVE, BID, and SID vulnerability identifiers are a subsumed property the system can know which hosts have a vulnerability, if it knows one of its ports has that same vulnerability.

The class membership rules are as follows:

[1] SNePSLOG is a logical language resembling higher-order logic that can be used as an interface to SNePS

1) all(x,y,z)({Isa(x,y),Ako(y,z)} &=> {Isa(x,z)}).

2) all(x,y,z)({Ako(x,y),Ako(y,z)} &=> {Ako(x,z)}).

Though the above rules aren't used by the current task, they are helpful in establishing a hierarchy of class member information, and will aid in future reasoning goals. The first rule asserts that if some entity is a member of some class (represented by the *Isa* predicate), and that class is a subclass, or a kind of, another class (represented by the *Ako* predicate), then that entity is a member of the superclass as well. The second rule provides a the transitivity rule to the *Ako* relationship.

3.2 Nessus

Nessus [4] is a system security tool that analyzes a specified set of hosts on a network for security vulnerabilities. Reports on these vulnerabilities are generated as an XML file, which details ports on the scanned hosts and CVE identifiers for vulnerabilities detected on those ports. Additional information can include the operating systems running on the hosts, network trace routes, host aliases, and severity of the vulnerability. An example of the Nessus output is presented in Fig. 1.

```
<host hostname="192.168.1.10">
  [...]
  <port portname="netbios-ns (137/tcp)">
    <alert>
      <hostname>192.168.1.10</hostname>
      <portname>netbios-ns (137/tcp)</portname>
      <id>10150</id>
      <level>NOTE</level>
      <desc>
        [...]
        CVE : CVE-2000-1194
      </desc>
    </alert>
  </port>
</host>
```

Figure 1: Sample Nessus output from our test network.

After generating a report we utilize a PERL program to parse the needed information from the file and represent it in the SNePSLOG syntax. These become facts in our knowledge base. The following proposition-valued terms are required to represent the

Nessus report:

- *PropertyValue(x,y,z)* - Entity *x* has a property *y* with value *z*
- *Isa(x,y)* - Entity *x* is a member of the category of *y*
- *PartOf(x,y)* - Object *x* is a part of object *y*
- *About(x,y)* - *x* contains information about *y*

As are the following terms:

- *Network* - the category of networks
- *Host* - the category of hosts.
- *Alert* - the category of alerts.
- *Active_Port* - the category of active ports.
- *CVE-xxxx-xxxx* - a specific CVE identifier where each *x* can be a distinct digit.
- *x.x.x.x* - a specific IP address where each *x* can be a number between 0 and 255.
- *low* - the low severity level, or what Nessus denotes with "NOTE"
- *tcp* - the tcp/ip protocol
- *Plugin_Id* - an identifier that uniquely identifies the structure of the information contained in an alert
- *Severity* – property that represents the severity level of an attack
- *IP_Address* - the IP address property
- *Number* - the port number property
- *Protocol* - the protocol property

The PERL script traverses the file looking for specific tags, and contents to generate SNePSLOG expressions. As it encounters tags we have determined to refer to certain entities (e.g. host, port, and alert) the PERL program creates a unique identifier for the entity. This identifier is merely the first letter of the entity in question followed by a number (e.g `p12' would refer to the twelfth port encountered. An example of the SNePSLOG expressions resulting from a parse of the Nessus data in Fig. 1 is pictured in Fig. 2. Note that Nessus doesn't explicitly have tags referencing the network, but the hosts in a Nessus file are a part of a network, so an identifier needs to be provided in order to form propositions about the scanned network. This identifier is `n1'.

```
Isa(n1, Network).
...
PartOf(h2, n1).
Isa(h2, Host).
PropertyValue(h2, IP_Address, 192.168.1.10).
...
Isa(p45, Active_Port).
PropertyValue(p45, Number, 137).
PropertyValue(p45, Protocol, tcp).
About(a96, p45).
Isa(a96, Alert).
PropertyValue(a96, Plugin_Id, 10150).
PropertyValue(a96, Severity, low).
...
PropertyValue(a96, CVE, CVE-2000-1194).
```

Figure 2: Sample SNePSLOG output from the Nessus parser

3.3 CVE Repository

CVE [2] is a database of common system vulnerabilities. It is available as an XML download, and is required for our purposes because it not only documents vulnerabilities across multiple operating systems, but also serves as a cross-reference for vulnerability referents. For example, the INFERD tracks use SIDs, which are translated to BIDs (Section 2.4), when identifying particular vulnerabilities, while Nessus uses CVE identifiers. Using this capability to cross reference CVE identifiers with BIDs, an INFERD track is determined to be a false-positive if the track's SID references a vulnerability whose CVE identifier doesn't reference that same vulnerability. This is a result of INFERD generating an attack track for a vulnerability the host doesn't possess, and thus, the track can be discarded.

This cross-referencing task is performed by the SNePS attached procedures facility. The size of the CVE repository prevents us from loading in all its information into a SNePS knowledge base without hindering the reasoning processes. Additionally, constantly consulting the web-site or XML file for entries would be computationally expensive. Thus, we use a PERL script to create a Lisp hash-table indexed on CVE identifiers, with values of BIDs and the ability of the attached procedures facility to integrate Lisp structures with the symbolic reasoning of SNePS.

Below is a sample of a few entries in the CVE database after being translated into Lisp hash-table code, which adds entries for every CVE-BID pair contained in the repository:

```
(setf (gethash 'CVE-1999-0002 *cve-table*) '(121 ))
(setf (gethash 'CVE-1999-0003 *cve-table*) '(122 ))
…
(setf (gethash 'CVE-1999-0842 *cve-table*) '(827 ))
(setf (gethash 'CVE-1999-0844 *cve-table*) '(823 820))
…
(setf (gethash 'CVE-2000-1194 *cve-table*) '(1227 ))
```

These entries are connected to SNePS through the proposition-valued term *CVE_BID_Equiv(cve,bid),* which is ultimately attached to the Lisp function specified in Fig. 3. For our purposes, we mostly rely on this function to determine if a CVE identifier refers to the same vulnerability as a given BID. Thus, *CVE_BID_Equiv(CVE-1999-0844, 820)* would return as known to the knowledge base, while *CVE_BID_Equiv(CVE-1999-0002, 1811)* would be unknown. This process allows our logical formalism to connect its symbolic representation with efficient Lisp functions in a similar manner to Prolog's arithmetic facilities. After the Lisp function terminates, the information is cached in the SNePS knowledge-base.

```
(define-attachedfunction cve-bid-equiv ((cve) (bid))
``If given two symbols, returns ``true'' if an entry exists in the hash-table
with a key of cve, and the list of corresponding bids contains bid. If given a
symbol for cve and variable for bid this will bind to the variable all the bids
corresponding to the cve"
 (cond
  ((and (symbolp cve) (integerp bid))
   (if (member bid (gethash (intern cve :snepslog) *cve-table*))
       `((snip:pos nil))
     `((snip:neg nil))))
  ((and (symbolp cve) (sneps:isvar.n bid))
   (if (first (gethash (intern cve :snepslog) *cve-table*))
       (loop for elm in (gethash (intern cve :snepslog) *cve-table*))
           collect (cons 'snip:pos  `(((,bid . ,elm)))))
     nil))
  (t nil)))
```

Figure 3: Attached function definition for cve-bid-equiv

3.3 SNORT Sensor Rules

SNORT Sensor rules [5] are used in a similar manner to the CVE repository. They are

structured text files that contain information that defines (possibly) malicious network packets or sequences of network packets. Without these rules, the Snort sensor would produce no alerts. When the rule is triggered by sensed network activity, the signature and unique SID associated with the rule is generated as information within the alert. We utilize these rules to establish a correlation between SIDs and BIDs, and ultimately SIDs and CVEs through the use of the CVE Repository (Section 2.3). Like the CVE parsing described previously, these files are parsed and used to create a Lisp hash-table keyed on SIDs with lists of corresponding BIDs as the value. An example of the results are as follows

```
(setf (gethash 494 *sid-bid-table*) '(1806 ))
(setf (gethash 497 *sid-bid-table*) '(1806 ))
…
(setf (gethash 1888 *sid-bugtraq-table*) '(5427 ))
(setf (gethash 1734 *sid-bugtraq-table*) '(10078 1227 1504 1690 4638 7307 8376 ))
```

These entries are connected to SNePS through the proposition-valued term *SID_BID_Equiv(cve,bid)*, which shares a similar implementation to the attached-procedure *CVE_BID_Equiv*, but on the *sid-bid-table*.

3.4 INFERD Attack Tracks

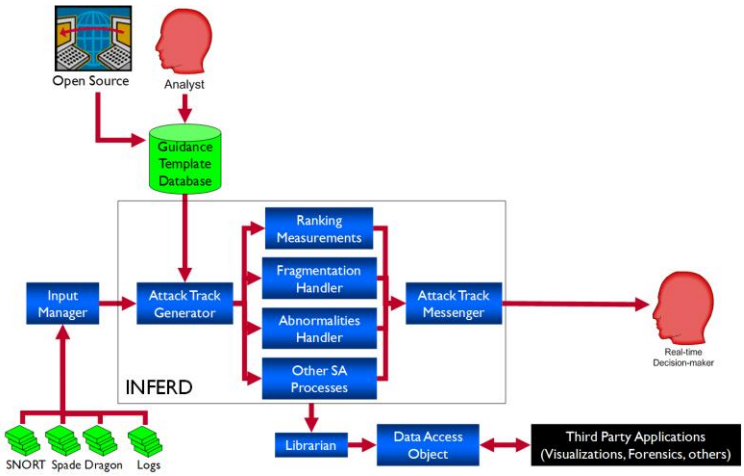

Figure 3 - INFERD Holistic Architecture

INFERD (Information Fusion Engine for Real-time Decision Making) is a perceptual stream-based fusion system whose genesis has been in the application area of cyber security. Figure 3 depicts the high level INFERD architecture. The fusion algorithms represented by the *Attack Track Generator* and *Other SA Processes* boxes in Figure 1 have been designed to be application independent. This allows the application of INFERD to heterogeneous problem environments through the development of new a priori models denoted as *Guidance Templates*.

The concept of operations (CONOPS) of INFERD within cyber security is to fuse

runtime alert information from various cyber sensors such as Snort and Dragon into tracks of attack activity. The attack track generation process instantiated to the cyber security problem through a cyber security Guidance Template produces attack tracks representing aggregated sequences of individual cyber attacks into INFERD's best estimate of a single multistage attack from a single attacker or attacking party.

This process has been designed to minimize and allow for incomplete or incorrect a priori information. This design detail has been found to be very important when considering the real world characteristics of cyber attacks. These attacks and the hacker attack methods are highly evolving and the problem of explicitly encoding an exhaustive set of multistage attack methods is not a tractable problem.

To maintain real-time alert processing performance in the very high frequency virtual environment of cyber security, tradeoffs had to be made. One such tradeoff was the decision to not model complex network topology information within INFERD. This information is required to evaluate false positives received from sensors, but complicates the problem with evolving network configurations and complex relationship analysis between attack signature and network configuration information.

The advantage of the SNePS / INFERD interface is that only the highest possible threat tracks can be evaluated via SNePS to be true positives which provide the complex reasoning processes performed within SNePS with the high speed aggregation capabilities performed within INFERD. Utilizing the Java-SNePS-API (Section 4.1) the SNePS system is queried with high threat INFERD attack track information to determine if the track or which components of the track are a false or true positive in terms of attack success.

An example fragment of an attack track is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Alert xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="AlertSchema.xsd">
…
<TargetIP>192.168.1.10</TargetIP>
…
<Sid>1734</Sid>
</Alert>
```

From this we extract the SID and Host, and assert the following information into the knowledge base: PropertyValue(*h*,SID,1734), where *h* is a host that matches the *TargetIP* tag value [e.g. *h* would be bound to 'h2', since the system knows *Isa(h2, Host)* and *PropertyValue(h2,IP_Address,192.168.1.10)*]. If no such host is known to SNePS, then the Nessus scan didn't include or found no vulnerabilities for *h*. With the above information in SNePS the system can be queried using *TruePositive(h,cve,sid).* Given the above track, the system would be queried using *TruePositive(h2,CVE-2000-1194,1734)* and return that it is the case that this track is a true-positive and is in need of further testing.

4 System Interfaces

4.1 Java-SNePS API

To facilitate a method of interacting with SNePS from INFERD a Java-SNePS API was developed. This interface is split into two processes, the SNePS process and the Java Virtual Machine (JVM), with a communication layer provided by Allegro Common Lisp's jLinker [3]. The Java-SNePS API provides access to the knowledge base through a set of tell-ask functions. These are defined as follows:

- *public static void tell (String s)* – Supplies string *s* to the SNePSLOG interpreter.
    - Ex. tell("PropertyValue(h2, IP_Address, 192.168.1.200).") will assert into the knowledge base the proposition *PropertyValue(h2,IPAddress,192.168.1.200)*

- *public static String[] ask (String s)* – Supplies string *s*, which must be a SNePSLOG or open proposition, to the SNePSLOG interpreter, and deduces all known instances of the supplied proposition. Results are returned as a list of strings representing the known propositions. A similar function, *askwh(s)* is also specified. It returns only the resulting variable bindings (if any variables are supplied) as a list of Lisp strings.
    - Ex. askwh("PropertyValue(h2, IP_Address, ?x)?") will return the IP address of *h2* as an array of one element: ["192.168.1.2"].

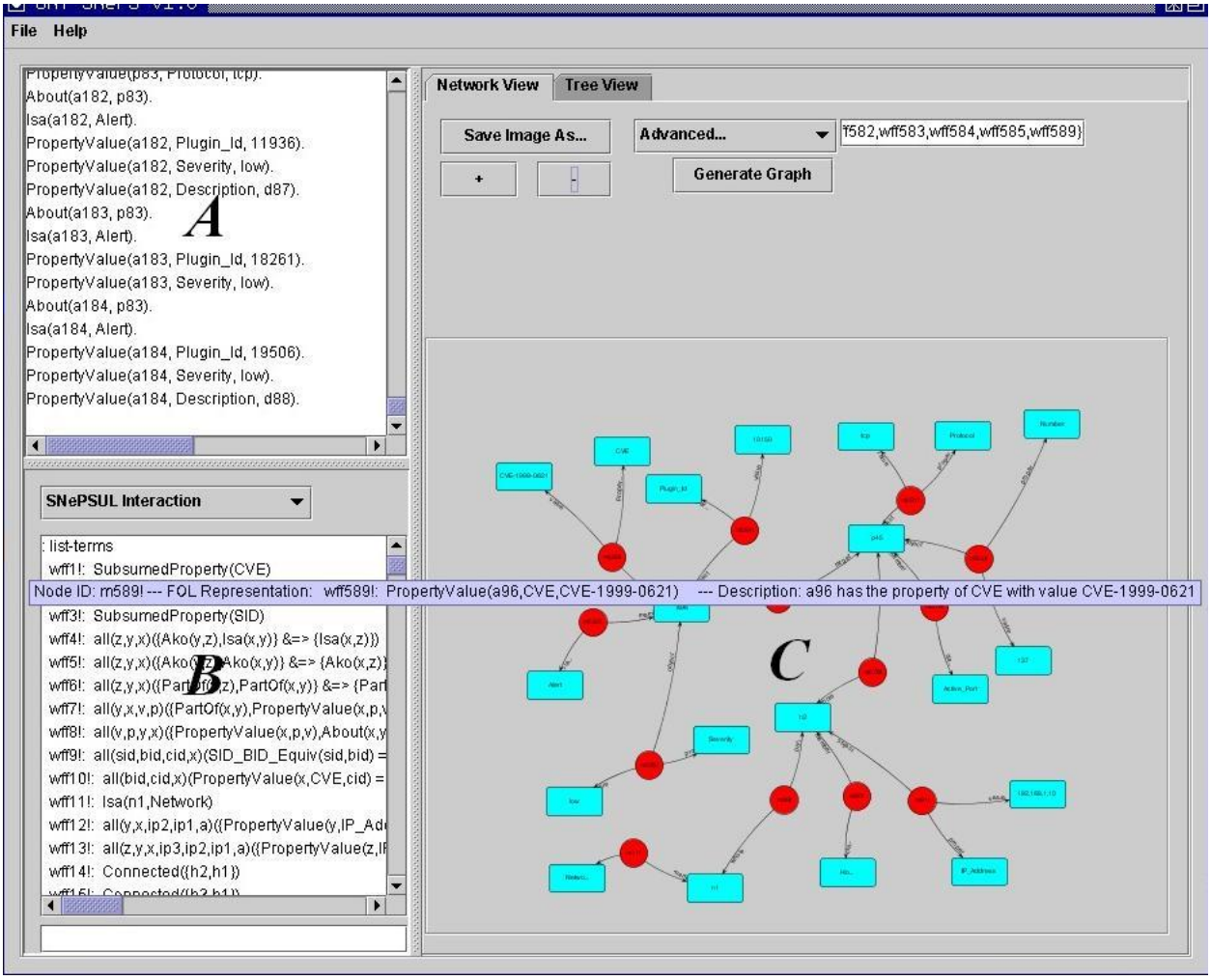

Figure 4 – SNePS GUI: (A) Loaded file, (B) SNePS Interaction pane, (C) Network View

4.2 SNePS GUI

Currently the SNePS GUI is capable of loading SNePSUL and SNePSLOG files, which are two different syntaxes for inputting data into SNePS. After loading a file the user can interact with the contents of the knowledge base through various views. Fig. 4 shows the GUI after loading our knowledge base, and displaying only the propositions shown in Fig. 2. The user can adjust the node positions, acquire information about them by hovering the cursor over the nodes, zoom in and out on the network, and save an image of the network to disk. Fig 4C demonstrates these capabilities. In the figure the user can enter the propositions they want displayed in the text box; or none at all, which will result in the entire network being displayed. Also depicted is the mouse highlighting a node, which causes the GUI to display information pertaining to the highlighted node. Fig. 5 shows a node image file saved from the GUI by using the "Save Image" button in Fig. 4C. In addition to the network view a user can also access information from the knowledge base using the standard SNePSLOG interactions through the input box in the lower left corner of the GUI (Fig. 4B). Finally, the user can view the binary proposition-valued terms in the GUI as a tree-hierarchy (Fig. 6). This hierarchy shows the *PartOf* relation (selected from the drop down menu), which shows the ports (expressed as the character *p* followed by a digit) that are part of a particular host (expressed as the

character *h* followed by a digit).

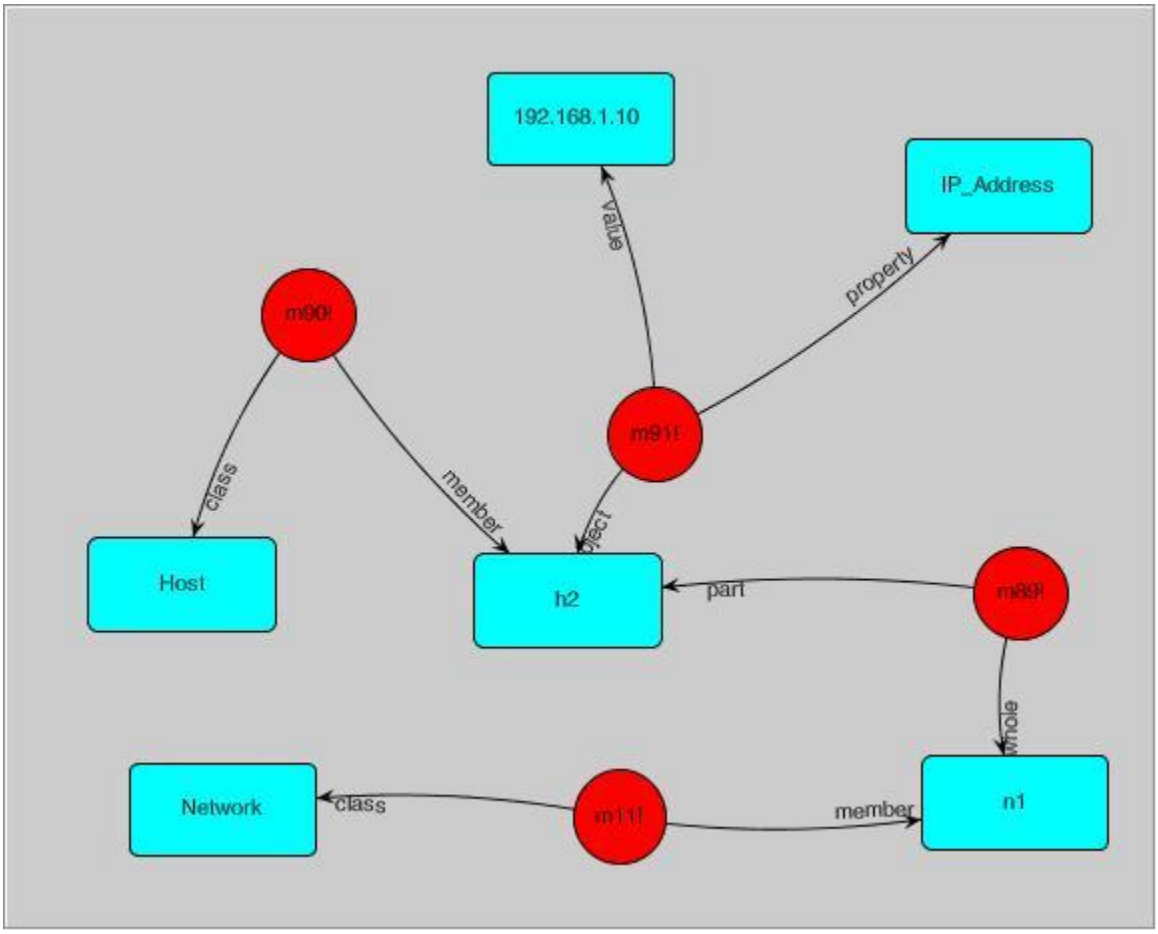

Figure 5 – Saved SNePS Network Image

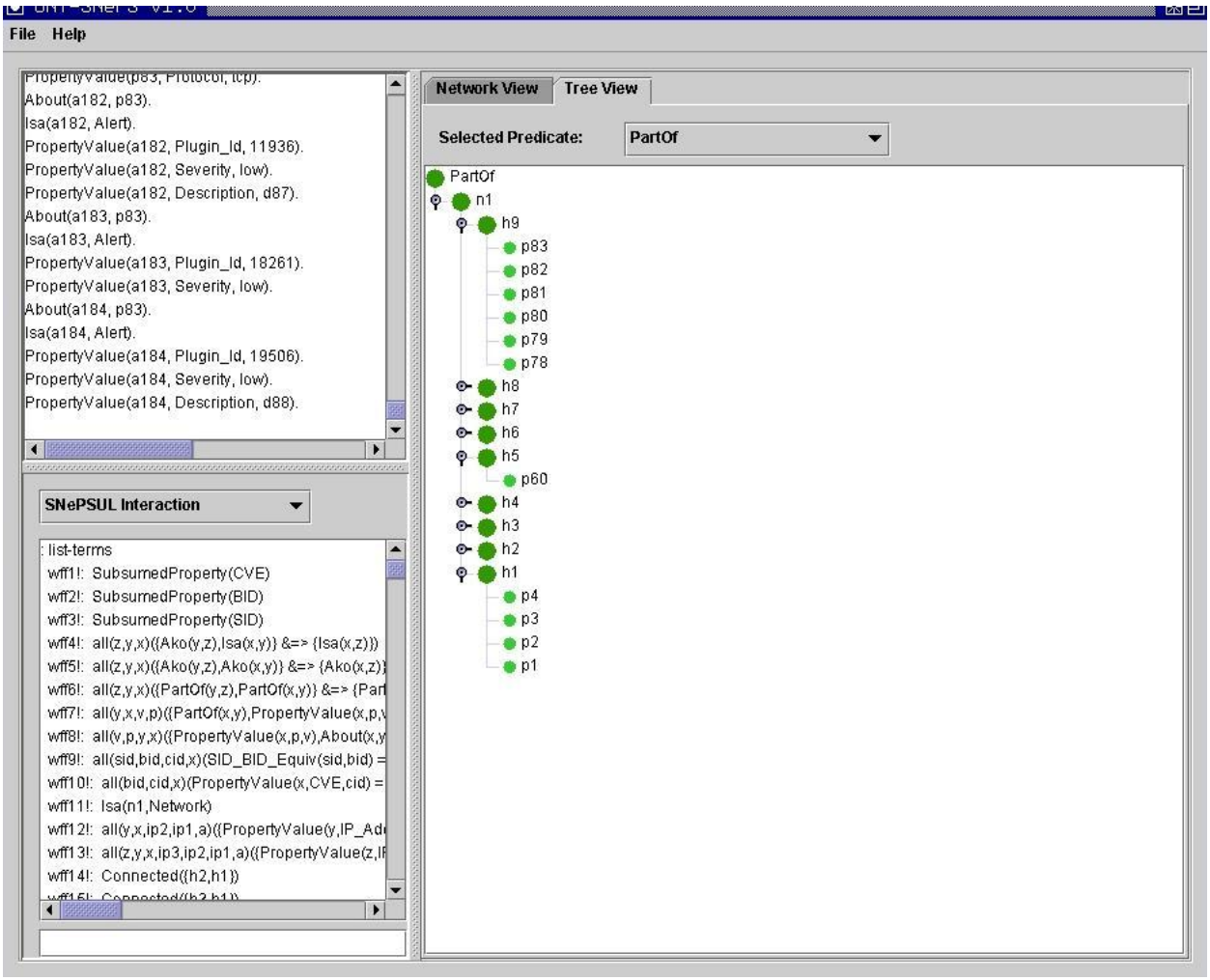

Figure 6 – SnePS GUI – Tree View

5 Future Work and Conclusions

This paper demonstrates our initial results from merging symbolic reasoning with cyber security. We have shown:

- A representation of system information in SNePS from a variety of disparate sources.
- A example of SNePS' reasoning system that reasoned about the represented information in order to determine false-positives in INFERD
- Methods of interfacing with SNePS, through both the Java-SNePS API and SNePS GUI
- New system features developed for the project, such as the SNePS Prolog-esque attached procedures facility.

Though initial results, the above allows us to build upon a working model for future goals. A variety of background knowledge still needs to be explored, in particular, a

typology of network devices that the Nessus scanning tool doesn't provide. More complex reasoning rules that capture the thought processes of the SME are also needed, in particular, we intend to explore the classification of hacker behavior and reasoning about firewall and intrusion detection system (IDS) settings in order to further classify INFERD attack tracks as false-positives. This latter process has been specified in design, but not implemented. It amounts to taking into account the network's topology and alerts generated by the IDS for a host potentially under attack. If the attack track generated by INFERD should not be rejected as a false-positive, all the IDS on route to the host in question, should generate an alert, if configured similarly. If they do not we reject the track as a false-positive. This type of scenario is depicted in Fig. 7. In this we would suspect an attack from the external hacker targeted for the host with IP 192.168.20.100 to generate alerts from the similarly configured IDS 192.168.2.1, 192.168.5.1, and 192.168.20.1.

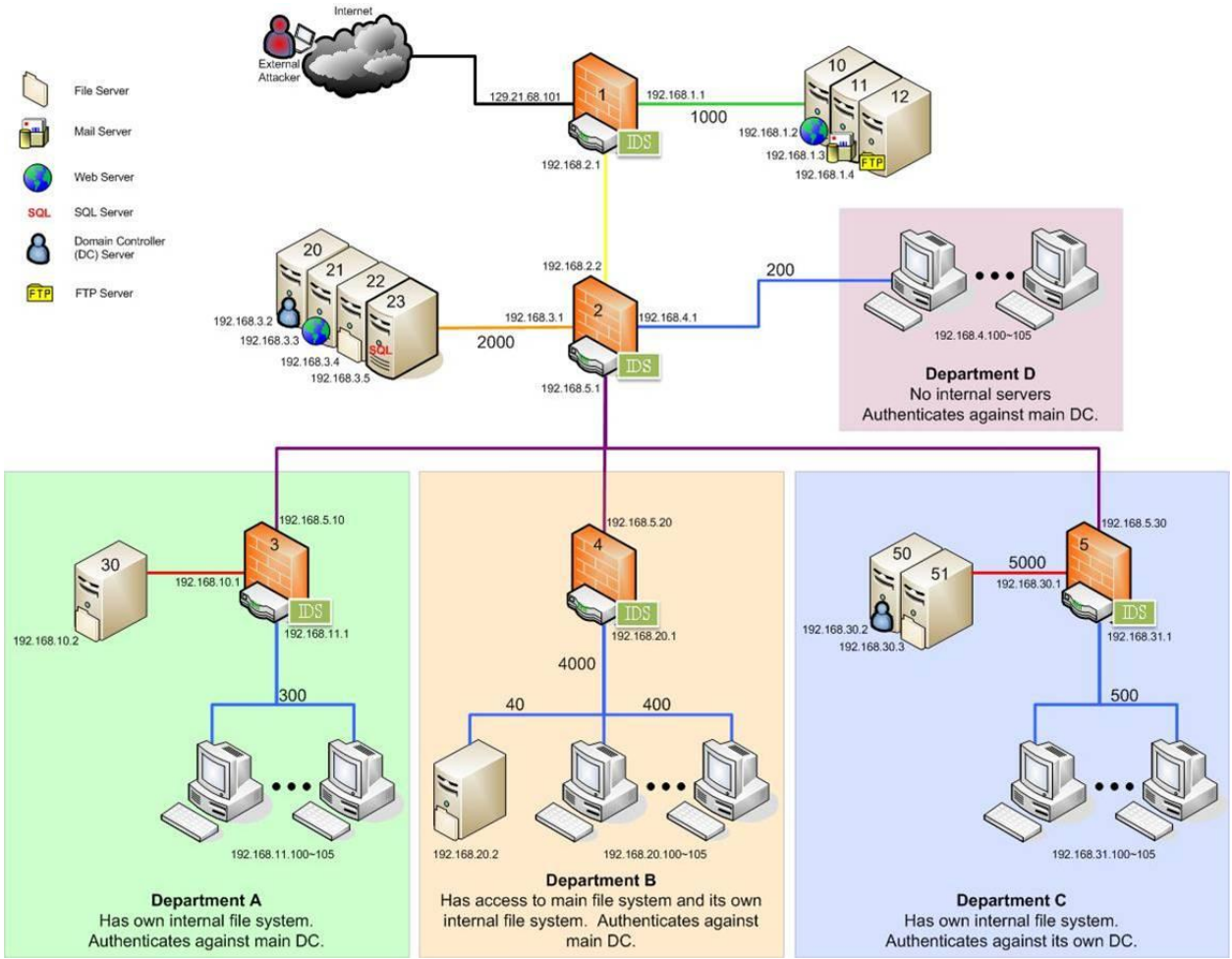

Figure 7 – Example Network Topology with multiple IDS on route to various hosts.
Figure courtesy of Dr. Jay Yang

Apart from symbolic reasoning, the GUI still requires better methods for manipulating the data, independent of the SNePS Interaction window. Though we have explored the OWL and RDF formats, a method of representing SNePS data in these formats would allow us to further our goals of having a flexible interface.

References

[1] CMIF Virtual Information Fusion Library (January 05, 2007). Center for MultiSource Information Fusion, School of Engineering & Applied Sciences, University at Buffalo. http://www.infofusion.buffalo.edu/

[2] CVE - Common Vulnerabilities and Exposures (December 07, 2006). The MITRE Corporation. http://cve.mitre.org/.

[3] jLinker – A Dynamic Link between Lisp and Java (April 30, 2007). Franz Inc. http://www.franz.com/support/documentation/8.0/doc/jlinker.htm

[4] Nessus (January 05, 2007). Tenable Network Security. http://www.nessus.org/.

[5] Snort – the de facto standard for intrusion detection/prevention (April 12, 2007). Sourcefire. http://www.snort.org/

[6] Stuart C. Shapiro and The SNePS Implementation Group, SNePS~2.6.1 User's Manual, Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY, October 6, 2004.

[7] A. Stotz and M. Sudit, "INformation Fusion Engine for Real-time Decision Making (INFERD): A Perceptual System for Cyber Attack Tracking", Proceedings of the 10th International conference on Information Fusion, July 2007.