The Use of SNePS for Cyber Security Reasoning* SNeRG Technical Note 44

Michael Kandefer, A. Patrice Seyed, and Stuart C. Shapiro
Department of Computer Science and Engineering
Center for Cognitive Science
National Center for Multisource Information Fusion
State University of New York at Buffalo
{mwk3|apseyed|shapiro}@cse.buffalo.edu

18th July 2008

1 Introduction

The National Center for Multisource Information Fusion (NCMIF) has endeavored to automate the process of detecting and handling an attack on a cyber network. Part of this process utilizes information fusion, a process of accumulating and refining data across disparate sources, to gather information from various network sources and provide an estimate of abnormal network activity. Such information is typically interpreted and acted upon by a cyber security Subject Matter Expert (SME). To at least partially automate the reasoning and acting processes of the SME, an automatic reasoner with acting is required, and for these purposes, we have chosen SNePS (Shapiro, 2000; Shapiro and The SNePS Implementation Group, 2008).

Much of our work has been discussed in (Kandefer et al., 2007). Though preliminary, it serves as a general statement of the goals of the project, and initial attempts at capturing the reasoning processes of a cyber security expert. The rest of this paper will discuss contents already found in the cited report, adding updates when necessary. Work done since that report will also be discussed.

*This work was supported in part by CUBRC under prime contract FA8750-06-C-0184 between CUBRC and U.S. Air Force Research Laboratory, Rome, NY.

2 Cyber Reasoning and SNePS

SNePS is a knowledge representation, reasoning, and acting system that is well suited for representing the reasoning processes of a SME. Some general capabilities that SNePS possesses that can aid in the process are:

- higher-order logical representation of facts and rules;
- backwards and forward inference;
- contradiction detection;
- acting;
- and procedural attachment.

Through the use of higher-order logic SNePS can represent any background knowledge or information sources the SME utilizes when reasoning about network abnormalities, and through the use of rules, any inferential processes needed to draw conclusions about that knowledge and data. However, this process does require translation, and, typically, the intercession of a knowledge engineer, an individual familiar with the process of representing information in logic, although graphical user interfaces (GUI) (such as the one discussed in (Kandefer et al., 2007)) and conversion tools from semantically defined formats (like OWL (W3C, 2004)) can help remove the knowledge engineer from the process.

SNePS employs a reasoning engine that is capable of performing backwards inference, a form of goal-directed or query driven reasoning, and forward inference, a process of deriving conclusions from new knowledge. Both are useful for reasoning in the cyber security domain. Backwards, because given the existing knowledge about the network and abnormalities one needs to query the reasoner about the abnormality (e.g., a query assessing whether the abnormality is an attack), and forward, because new information is constantly being processed by the reasoner. As part of its reasoning engine, SNePS also can detect contradictions made when reasoning or adding new information to the knowledge base, and then correct the issue with user input.

To help capture the ability of an SME to handle attacks when they are detected, a method for representing and executing planned acts is necessary. SNePS processes an acting system that is integrated with the reasoning system (Kumar and Shapiro, 1994). With this ability a variety of conditional plans can be represented, such that, when the system reasons to some conclusion it can act on the results appropriately (e.g., when an abnormality is detected and reasoned to be an attack, then execute a procedure for notifying the users of the system).

Finally, procedural attachment is available in SNePS for efficient computation of mathematical predicates, or for performing look-ups in huge data repositories, a feature necessary for reasoning in the cyber security domain, which relies on vast vulnerability databases.

3 Example of Cyber Reasoning

An "attack track" is a report that alleges that some network device, referred to as the "target," was attacked, that the attack originated from some "source" device, and that it exploited some vulnerability of the target. The vulnerability is identified by a "signature identifier" (SID). There is additional information in the attack track that is not relevant to the discussion in this paper. One problem is that some attack tracks report attacks that didn't really happen, or were incidents that did not really constitute attacks. SNePS is being used to try to identify such "false positives."

Various procedures and knowledge required for identifying attack tracks as false positives were elicited from an SME. As a result, two forms of reasoning were chosen for implementation.

3.1 Vulnerability Correspondence

The first form of reasoning elicited from the SME involves determining whether the SID of the attack track indeed corresponds with a known vulnerability of the network. The network vulnerabilities are provided as "terrain data," which are translated into SNePS by an automatic interpreter written for this purpose. If the SID does not correspond to any vulnerability, then the attack is deemed improbable, and the attack track is flagged as a false positive.

Unfortunately, the network terrain data give the vulnerabilities of the network hosts as Common Vulnerabilities and Exposure (CVE) (The MITRE Corporation., 2006) identifiers, which are different from the SIDs reported in the attack tracks. (The terrain data also include exposure vulnerabilities for hosts, which are basically the English names of the exposures, and don't use a particular identification scheme like CVE or SID.) There is a CVE repository that provides a correspondence between CVEs and Bugtraq identifiers (BIDs), a third form of vulnerability labeling. Finally, files of SNORT (Sourcefire, 2007) sensor rules provide a correspondencs between BIDs and SIDs.

This technique of identifying false positives is:

1. The attack track, *a* reports that the target device *n* was attacked via its vulnerability *sid*.
2. Use the terrain data to find the CVE identifiers of vulnerabilities that *n* is subject to.
3. Look up the CVE identifiers in the CVE repository to find their corresponding BIDs.
4. Use the SNORT file to determine if any of those BIDs corresponds to *sid*.
5. If not, then *a* is a false positive.

The SID-BID-CVE connection is represented in SNePS with the following two predicates.

- `CVE_BID_Equiv(`*cve*`,`*bid*`)`
  CVE *cve* denotes the same vulnerability as BID *bid*. (From the CVE repository.)
- `SID_BID_Equiv(`*sid*`,`*bid*`)`
  SID *sid* denotes the same vulnerability as BID *bid*. (From the SNORT files.)

All the instances of these relations are not stored in the SNePS knowledge base ahead of time. Instead, procedural attachment is used to look up the required information from the appropriate repository when needed.

In addition, the following SNePS predicates are also used.

- `PropertyValue(`*`n,p,v`*`)`
  The property *p* (either `CVE` or `BID`) of network device *n* is *v*.
- `NetworkDevice(`*`n,nid,nip`*`)`
  The network id of network device *n* is *nid* and its IP address is *nip*.
- `Alert(`*`a,ip,sid`*`)`
  Attack track *a* reports that the target device that has the IP address *ip* was attacked via a vulnerability with the SID *sid*.
- `CVE(`*`cinst,cid`*`)`
  cinst is an instance of the CVE with the id *cid*.
- `PartOf(`*`v,n`*`)`
  Network device *n* has the particular vulnerability or exposure *v*.
- `PossibleAttackSID(`*`a,sid`*`)`
  Attack track *a* reported an attack via a vulnerability *sid* that exists on the specified host.

Using these predicates, the rules for determining that an attack track is a false positive are:

- If a network device, *ninst* has a particular instance of a vulnerability type with identifier *cid*, then we say that *ninst*'s CVE is *cid*.

```
all(ninst,nid,nip)(NetworkDevice(ninst,nid,nip)
                   => all(cinst,cid)(CVE(cinst,cid)
                          => (PartOf(cinst,ninst)
                              => PropertyValue(ninst,CVE,cid))))
```

- A network device has the BID *bid* that corresponds to every CVE id that it has.

```
all(ninst,nid,nip)(NetworkDevice(ninst,nid,nip)
          => all(cid)(PropertyValue(ninst,CVE,cid)
                      => all(bid)(CVE_BID_Equiv(cid,bid)
                                  => PropertyValue(ninst,BID,bid))))
```

- An attack track is possible if its SID corresponds to any BID (or CVE) its alleged target has.

```
all(a,sid,nip)(Alert(a,nip,sid)
          => all(n,nid)(NetworkDevice(n,nid,nip)
                        => all(bid)(PropertyValue(n,BID,bid)
                                    => (SID_BID_Equiv(sid,bid)
                                        => {PossibleAttackSID(a,sid)})))).
```

3.2 Firewall Rules

Even if the target device does have the vulnerability which the attack track reported that the source exploited, the attack track could be a false positive if there was no way for the source to communicate with the target. This is the second form of reasoning elicited from the SME, and uses connectivity rules for the network—what network devices are allowed access to which others on specific ports. These rules are supplied by the network terrain data in addition to the vulnerability information. The basic relation is represented by the SNePS predicate,

- `ConnectedByPort(`*`src`*`,`*`dst`*`,`*`port`*`,`*`prot`*`)`
  The source network device, *`src`*, is connected to the destination network device, *`dst`*, on port *`port`* using protocol *`prot`*.

This relation is not directly supplied by the network terrain data, but can be inferred from the data that is supplied. (See (Seyed et al., 2008, §3.11) for the rule that is used.) Nevertheless, it still only specifies network devices that are directly connected. Clearly, devices can be indirectly connected. Rather than using standard logical inference to derive these indirect connections, we use SNePS's "path-based inference". A SNePS proposition is represented by a slot-filler frame, where the slots specify the argument positions and the fillers are the arguments in those positions. For example, the proposition

```
ConnectedByPort(nd1,nd2,p1,tcp)
```

is represented by the frame

```
(src (nd1) dst (nd2) port (p1) prot (tcp))
```

A SNePS knowledge base can also be viewed as a directed labeled graph, in which each frame is a node and each slot labels a directed arc from the frame it is in to each of the fillers. Path-based inference involves inferring an arc labelled *`r`* from a node *`n`* to a node *`m`* whenever a certain path goes from *`n`* to *`m`* (Shapiro, 1991). Following paths in a SNePS network is more efficient than using normal logical inference, but is not always applicable. It is particularly appropriate, however, for reasoning about transitive relations, which connectedness is, being the transitive closure of `ConnectedByPort`.

To specify that the presence of an arc may be inferred from the presence of a path, a path-based inference rule is specified. We will not explain the syntax of path-based inference rules here. (See (Shapiro and The SNePS Implementation Group, 2008, §2.5.2) for the details.) However, the rules for the relations `src` and `dst` are:

- If *a* has *b* as a src, and *b* has *c* as a src, then *a* has *c* as a src.
  If *a* does not have *c* as a src, but *b* has *c* as a src, then *a* does not have *b* as a src.

```
define-path src
            (or src
                (compose ! src (kstar (compose dst- ! src)))
                (domain-restrict ((compose arg- ! max) 0)
                                 (compose src
                                          (kstar (compose src- ! dst)))))
```

- If *a* has *b* as a dst, and *b* has *c* as a dst, then *a* has *c* as a dst. If *a* does not have *c* as a dst, but *b* has *c* as a dst, then *a* does not have *b* as a dst.

```
define-path dst
            (or dst
                (compose ! dst (kstar (compose src- ! dst)))
                (domain-restrict ((compose arg- ! max) 0)
                                 (compose dst
                                          (kstar (compose dst- ! src)))))
```

A key part of both these rules is the path constructor `(kstar p)`, which means zero or more occurrences of the path *p*, and allows an arc to be implied by a path of arbitrary length.

3.3 Using SNePS Reasoning

With the above rules and background knowledge in place SNePS can be "plugged into" the automatic cyber security management system. This is handled through the use of a SNePS executable and a Java class that provides `tell/ask` methods for interfacing with the executable. Both the executable and Java interface were developed for this project utilizing the ACL JLinker libraries (Franz Inc., 2008) and executable generator. The Java interface provides two methods to the overall system:

- `public InferdSnepsAPI(String snepsExePath, int interfacePort)`

  A constructor that starts and returns a connection to the SNePS executable.

- `public double doFalsePosCheck(Document doc)`

  Returns a double value that represents a false positive measure for the given attack track doc. Currently, the possible values are 0.5 or 0.0, the former indicating a false positive, the latter indicating that a false positive couldn't be determined.

The most important method is `doFalsePosCheck`, which works by asserting attack track information into SNePS, and then querying the system based on the new information. The algorithm is as follows.

1. Given attack track *a* with:
    - Source IP: *sip*
    - Target IP: *tip*
    - Port: *port*
    - Protocol: *prot*
    - SID: *sid*
    - Exposure: *exp*
2. Assert into SNePS the alert information, `Alert(a,tip,sid)`.
3. Query SNePS to determine if it contains any information about the network devices with those identifiers (e.g., `NetworkDevice(?n,?nid,sip)?`), the exposure (e.g. `Exposure(?e, exp)?`), or if the exposure is known to be on the host in question (e.g., `PartOf(eid,tid)?`, where *eid* and *tid* are the identifiers for the exposure and target network device in the knowledge base). If not, return 0.5.
4. Query SNePS to determine if the target host has the vulnerability specified (e.g., `PossibleAttackSID(a,sid)?`). If not, return 0.5.
5. Query SNePS to determine if the two devices are connected (e.g., `ConnectedByPort(sid,tid,port,prot)?`, where *sid* and *tid* are the network identifiers for these devices. This information is acquired from the query made in step (2)). If not, return 0.5.
6. Return 0.0.

4 Results

The algorithm was tested on 6 cases, one test for each possible output. All outputs were as expected given the input files. The reasoner was deemed to be unacceptably slow when running with the rest of the system on a Windows desktop computer, but ran significantly faster on a Department of Computer Science and Engineering computer, nickelback, a Sun Sunfire X4200 with 8.0 GB of main memory and two Dual Core AMD Opteron^{TM} Processor 285s, clocked at 2592 MHz, running RedHat Enterprise Linux 4 (64-bit).

Following are outputs generated by the 6 runs of the system (each "_*i*xml file contains an attack track).

1. Enter an alert file to parse: _1.xml
   Performing false pos check on:
   Alert[source ip: 131.46.41.51
         target ip: 192.168.1.200
         protocol: tcp
         port: 445
         signature: NETBIOS SMB-DS IPC$ unicode share access
         sid: 2559]
   Determining if this host is in the virtual terrain...
   False positive: No target host found with specified IP:192.168.1.200

2. Enter an alert file to parse: _2.xml
   Performing false pos check on:
   Alert[source ip: 100.10.20.4
         target ip: 192.168.1.2
         protocol: icmp
         port: unknown
         signature: ICMP L3retriever Ping
         sid: 466]
   Determining if this host is in the virtual terrain...
   Determining if the signature is on any host in the virtual terrain...
   False positive: Exposure(ICMP L3retriever Ping) does not reference an
   exposure in the knowledge base. No host is known to have it.

3. Enter an alert file to parse: _3.xml
   Performing false pos check on:
   Alert[source ip: 192.168.1.3
         target ip: 192.168.10.2
         protocol: tcp
         port: 445
         signature: MS-SQL xp_showcolv possible buffer overflow
         sid: 2466]
   Determining if this host is in the virtual terrain...
   Determining if the signature is on any host in the virtual terrain...
   Determining if the signature corresponds to a vulnerability on this host...
   False positive: Exposure(MS-SQL xp_showcolv possible buffer overflow)
   does not correspond to a vulnerability on host 192.168.10.2

4. Enter an alert file to parse: _4.xml
   Performing false pos check on:
   Alert[source ip: 100.10.20.9
         target ip: 192.168.1.3
         protocol: tcp
         port: 25
         signature: SMTP RCPT TO overflow
         sid: 252]
   Determining if this host is in the virtual terrain...
   Determining if the signature is on any host in the virtual terrain...
   Determining if the signature corresponds to a vulnerability on this host...
   Determining if SID corresponds to a CVE vulnerability...
   False Positive: SID 252 does not correspond to a CVE vulnerability on
   192.168.1.3

5. Enter an alert file to parse: _11.xml
   Performing false pos check on:
   Alert[source ip: 100.10.20.9
         target ip: 192.168.1.2
         protocol: icmp
         port: unknown
         signature: WEB-MISC bad HTTP/1.1 request
         sid: 1650]
   Determining if this host is in the virtual terrain...
   Determining if the signature is on any host in the virtual terrain...
   Determining if the signature corresponds to a vulnerability on this host...
   Determining if SID corresponds to a CVE vulnerability...
   Determining if a connection is possible between the source and target,
   according to firewall rules...
   False positive: 100.10.20.9 and 192.168.1.2 are not connected.

6. Enter an alert file to parse: _6.xml
   Performing false pos check on:
   Alert[source ip: 100.10.20.9
         target ip: 192.168.1.2
         protocol: unknown
         port: unknown
         signature: WEB-MISC bad HTTP/1.1 request
         sid: 1650]
   Determining if this host is in the virtual terrain...
   Determining if the signature is on any host in the virtual terrain...
   Determining if the signature corresponds to a vulnerability on this host...
   Determining if SID corresponds to a CVE vulnerability...
   Determining if a connection is possible between the source and target,
   according to firewall rules...
   SNePS could not determine a false positive for this attack track.

5 Addressing Criticisms

There have been two major criticism levied against the approach discussed above: that SNePS does not provide a service significantly different from that provided by database systems, expert system shells, or ontology reasoners; that SNePS is too slow to be useful.

In the cyber security architecture, SNePS serves as a false positive arbiter on the attack tracks generated by the network sensors. The two techniques for determining false positives elicited from the SME did not make use of the expressiveness available in SNePS, which has a much more expressive language than that which is typical of many database systems, expert systems, or ontology reasoners (see (Shapiro and The SNePS Implementation Group, 2008) and (Kandefer and Shapiro, 2008)).

The SNePS reasoner was deemed to be too slow when running as part of the overall system on a Windows desktop computer. SNePS was considered important enough to assign us the task of finding some of the reasons for its slowness, and improving it. The success of that task is reported in (Seyed et al., 2008).

6 Conclusions

SNePS is simultaneously a logic-based, frame-based, and network-based knowledge representation, reasoning and acting system. Its logic-based aspect supports logical reasoning; its frame-based aspect supports slot-based reasoning (which wasn't utilized in this project); and its network-based aspect supports path-based reasoning. Procedural attachment may be used so that, when instances of certain predicates are required, they can be retrieved from large external data bases or files written in XML or other formats. Although implemented in Common Lisp, SNePS has an API implemented in Java, so that it can can be used in a large system with other software packages implemented in other languages. SNePS's ability to interact with programs written in a variety of languages, its ability to use data contained in a variety of file formats, the expressiveness of its knowledge format, and the variety of its reasoning methods make it a useful and valuable component of information fusion systems.

In this project, we used SNePS to represent and reason about network information, attack tracks, and the information in vulnerability databases to aid in the maintenance of cyber security. We elicited and implemented reasoning strategies from a Subject Matter Expert regarding vulnerabilities and firewall settings relevant to the assessment of an abnormality as a network threat or a false positive, and demonstrated the success of this approach.

References

Franz Inc. (2008). jLinker - a dynamic link between Lisp and Java. http://www.franz.com/support/documentation/8.1/doc/jlinker.htm.

Kandefer, M., Shapiro, S., Stotz, A., and Sudit, M. (2007). Symbolic reasoning in the cyber security domain. In *Proceedings of MSS 2007 National Symposium on Sensor and Data Fusion*.

Kandefer, M. and Shapiro, S. C. (2008). Comparing SNePS with Topbraid/Pellet. SNeRG Technical Note 42, Department of Computer Science and Engineering, The State University of New York at Buffalo, Buffalo, NY.

Kumar, D. and Shapiro, S. C. (1994). Acting in service of inference (and *vice versa*). In Dankel II, D. D., editor, *Proceedings of The Seventh Florida AI Research Symposium (FLAIRS 94)*, pages 207–211. The Florida AI Research Society.

Seyed, A. P., Kandefer, M., and Shapiro, S. C. (2008). SNePS efficiency report. SNeRG Technical Note 43, Department of Computer Science and Engineering, The State University of New York at Buffalo, Buffalo, NY.

Shapiro, S. C. (1991). Cables, paths and "subconscious" reasoning in propositional semantic networks. In Sowa, J., editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 137–156. Morgan Kaufmann, Los Altos, CA.

Shapiro, S. C. (2000). SNePS: A logic for natural language understanding and commonsense reasoning. In Iwańska, Ł. M. and Shapiro, S. C., editors, *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*, pages 175–195. AAAI Press/The MIT Press, Menlo Park, CA.

Shapiro, S. C. and The SNePS Implementation Group (2008). *SNePS 2.7 User's Manual*. Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY. Available as http://www.cse.buffalo.edu/sneps/Manuals/manual27.pdf.

Sourcefire (2007). Snort: the de facto standard for intrusion detection/prevention. http://www.snort.org/.

The MITRE Corporation. (2006). CVE - Common Vulnerabilities and Exposures. http://cve.mitre.org/.

W3C (2004). OWL Web Ontology Language Overview. http://www.w3.org/TR/2004/REC-owl-features-20040210/.