Natural Language Interface to an Intrusion Detection System

K. Wee[1,*] , T. Collier[2,**], G. Kobele[2,***], E. Stabler[2,****], and C. Taylor[2,*****]

* Graduate School of Information and Communication, Ajou University, Suwon, Korea 442-749
(Tel : 82-31-219-2635; Fax : 82-31-219-1811 ; E-mail: kbwee@ajou.ac.kr)
** Department of Organismic Biology, Ecology, and Evolution, University of California, Los Angeles, CA 90095
(E-mail: travc@alife.org)
*** Department of Linguistics. University of California, Los Angeles, CA 90095
(E-mail: kobele@ucla.edu)
**** Department of Linguistics. University of California, Los Angeles, CA 90095
(E-mail: stabler@ucla.edu)
***** Department of Organismic Biology, Ecology, and Evolution, University of California, Los Angeles, CA 90095
(E-mail: taylor@biology.ucla.edu)

Abstract: Computer security is a very important issue these days. Computer viruses, worms, Trojan horses, and cracking are prevalent and causing serious damages. There are also many ways developed to defend against such attacks including cryptography and firewalls. However, it is not possible to guarantee complete security of computer systems or networks. Recently much attention has been directed to ways to detect intrusions and recover from damages. Although there have been a lot of research efforts to develop efficient intrusion detection systems, little has been done to facilitate the interaction between intrusion detection systems and users. Reports and presentations of current states of the computers or networks to the user in a format that is easy to understand as well as specification of security policies from the user are important aspects of intrusion detection systems. We present our first steps to develop natural language interface between a user and an intrusion detection system using minimalist transformational grammar and Prolog. Our system takes a pseudo-English query from the user, parses it using CYK-like algorithm, converts it into a formula in Horn logic, feeds it to Prolog, and then gets the answer back in a simple format.

Keywords: intrusion detection, computer security, natural language interface, minimalist grammar, Prolog

1. Introduction

Computer security has become an important issue and is ever more critical as computers are rapidly and densely interconnected worldwide. Any novice equipped with minimal knowledge of computing can breach computer security because of the wide availability of hacking scripts and tools. Furthermore, current operating systems are so complex they cannot practically be made provably secure from attack.

There are many ways to enhance the security of computer systems: encryption of data, access control, programs that detect malicious code such as computer viruses and Trojan horses, and so on. One such method that is receiving attention is the use of an intrusion detection system (IDS). These attempt to identify, preferably in real time, unauthorized use, misuse, and abuse of computer systems and networks [3].

IDSs can be categorized according to the detection methods employed: misuse detection, anomaly detection, and specification-based detection. A *misuse detection model* recognizes known patterns of exploitations and guards against attack by matching the current events to those known patterns. The principal shortcoming of this approach is that it cannot detect previously unknown or unsuspected means of attack. *Anomaly detection models* monitor and learn normal behaviors of the computer systems or networks, and identify suspicious behaviors that deviate from the norm. These may use a variety of techniques to learn normal behavior: statistics, finite state automata, hidden Markov models, neural networks, genetic algorithms, and artificial immune systems have all been employed [4,5]. Problems with anomaly detection methods are that they have high probabilities of issuing false alarms, and, at least in theory, that intruders could train the IDS in such a way that their attacks are considered normal behavior. *Specification-based models*

[1] This work was supported by Brain Korea 21 project.
[2] This work was supported by NSF LIS program.

formally specify acceptable behaviors of programs and report behavior that does not conform to the specifications[7,8]. In theory these have zero rate of false alarm, but at the cost of having to specify normal behavior of every program we want to monitor. Commercial packages typically employ a combination of these techniques.

While much effort has been put into developing new techniques for detecting intrusions, less attention has been paid to their user interfaces[1]. The interface between the IDS and its user should present only the relevant information to the user in a way that is easy to understand in order to facilitate monitoring and controlling the IDS. The user in turn should be able to specify the policy in a way that the IDS can enforce. It would be helpful to have mechanisms for making specific queries to the IDS and to get just the relevant information back. When the IDS reports a potential intrusion alarm, the user would like to have information of the current states of the computer system available in a format that will enable the user to determine whether it is really an intrusion, how the intrusion was staged, and what damage was incurred.

In this paper, we present our first steps toward developing a system whereby the IDS and its user can interact using natural language. Section 2 describes the architecture of our system, Section3 explains the experiments we performed. Section 4 concludes the paper.

2. Architecture

Our system takes queries in English, parses the sentence using a minimalist transformational grammar[2,9,10], converts it into predicate logic expressions, feeds it to Prolog, and gets an answer back in a simple format. Chomsky[2] notes that although finite inflection appears on English verbs, when one considers the inflectional patterns in auxiliary verb sequences, negated sentences, and adverb positions, it is natural to assume that the tense marking is in some sense “above” the verb, close to the front of the sentence. In the same spirit, agreement, complementizers, and determiners do not intervene between lexical categories, but wrap around them from above.

We designed and implemented the prototype of such a system on Red Hat Linux (version 7.0) using Sicstus Prolog. We are exploring a score of predicates including *start*(Subject, Program), *chmod*(Subject, File, Permission), *open*(Subject, File, Mode), *exec*(Subject, Program), *fork*(Subject, PID), and a few others. When an English query is entered, it is parsed by a CYK-like parser[6], fed to Prolog, and then the answer is returned by Prolog (figure 1). Predicates are imp lemented by use of system predicates in Sicstus Prolog, various filters, and audit trails generated by a Linux auditing system called *Seer Observer* developed by Geoff Kuenning of the File Mobility Group in the UCLA Computer Science Department.

The following examples show the trace of processing a query and the corresponding minimalist grammar.

```
|: who open ed /etc/passwd?
building chart..............::accepted
as category c

%%% Sentence parsed:
who open ed /etc/passwd

%%% Compact representation:
2 4 5 14 29 35

%%% Standard semantic representation:
[t:[question,[past,[[tr,[open_meaning,
/etc/passwd]],wh]]]]
```

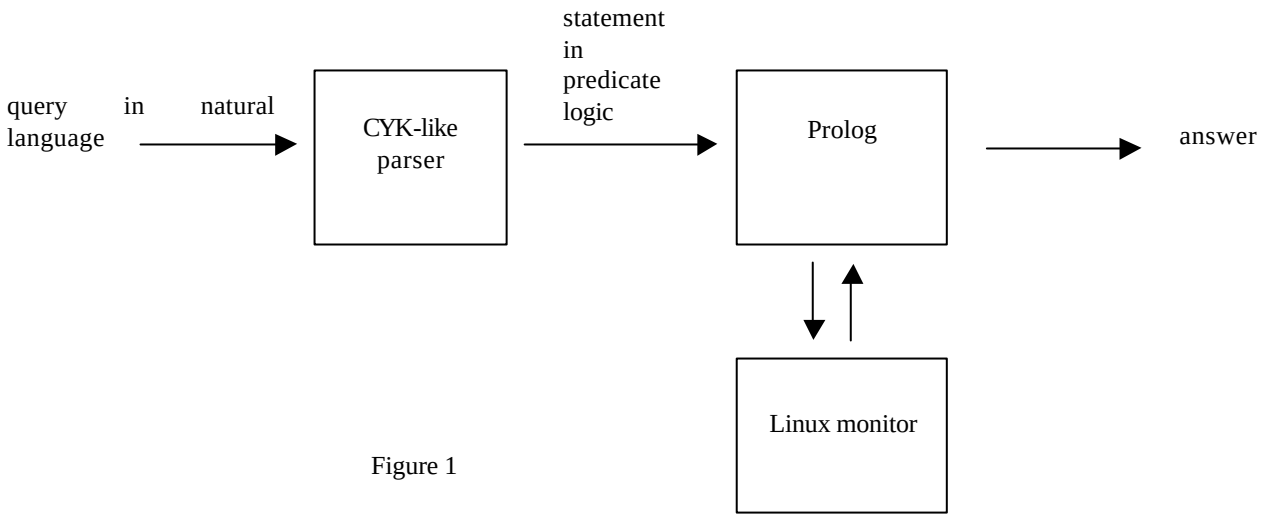

Figure 1

```
%%% Logic:
~wh(A,some(B,some(C,open_meaning(A,/etc/passwd,B,C) and
      C<<datime(2001,7,20,16,5,37))))

%%% CNF:
(-A<<datime(2001,7,20,16,5,37) v
     -open_meaning(B,/etc/passwd,C,A))

%%% Prolog:
answer(A) :-
usr:B<<datime(2001,7,20,16,5,37),
usr:open_meaning(A, /etc/passwd',_, B).

%%%
actor([31345,??,'500']),actor([32147,'/bin/vi','500'])].


% complementizers, marking assertive force
[]::[=i,c]@((t->t):assert).
[]::[=i,+wh,c]@((t->t):question).

% inflectional elements, marking tense
[s]::[=pred,+v,+k,i]@((t->t):pres).
[ed]::[=pred,+v,+k,i]@((t->t):past).

% transitiviser tr
[]::[=vt,+k,=d,pred]@(((e->t)->(e->t)):tr).
[]::[=v,pred]@((t->t):id).

% verbs
[praise]::[=d,vt,-v]@((e->e->t):praise).
[laugh]::[=d,v,-v]@((e->t):laugh).
[be]::[=d,vt,-v]@((e->e->t):be).

% audit verbs
[create]::[=d,vt,-v]@((e->e->t):create).
[open]::[=d, vt, -v]
@((e->e->t):open_meaning).

% names
['Lavinia']::[d,-k]@(e:'Lavinia').
['Titus']::[d,-k]@(e:'Titus').
[root]::[d,-k]@(e:actor([_PID,_ProcessName,'0'])).
[travc]::[d,-k]@(e:actor([_PID,_ProcessName,'500'])).

% files
[cdrom]::[d,-k]@(e:cdrom).
['/etc/passwd']::[d,-k]@(e:'/etc/passwd').

% processes
[cp]::[d,-k]@(e:actor([_PID,'/bin/cp',_UID])).
[vi]::[d,-k]@(e:actor([_PID,'/bin/vi',_UID])).

% wh-names
[who]::[d,-k,-wh]@(e:wh).
[what]::[d,-k,-wh]@(e:wh).

% determiners
[some]::[=qual,d,-k]@(((e->t)->(e->t)->t):some).
[every]::[=qual,d,-k]@(((e->t)->(e->t)->t):every).
[a]::[=qual,d,-k]@(((e->t)->(e->t)->t):some).
[an]::[=qual,d,-k]@(((e->t)->(e->t)->t):some).

% wh-determiners
[which]::[=qual,d,-k,-wh]@(((e->t)->(e->t)->t):wh).
[what]::[=qual,d,-k,-wh]@(((e->t)->(e->t)->t):wh).

% adjectives of size
[big]::[=nat,size]@(((e->t)->(e->t)):big).
[little]::[=nat,size]@(((e->t)->(e->t)):little).
[]::[=nat,size]@((_->_):id).

% adjectives of nationality
[unix]::[=n,nat]@(((e->t)->(e->t)):unix).
[windows]::[=n,nat]@(((e->t)->(e->t)):windows).
[]::[=n,nat]@((_->_):id).

% nouns
[file]::[n]@((e->t):file).
[process]::[n]@((e->t):process).
[action]::[n]@((e->t):action).
[human]::[n]@((e->t):human).
```

3. Experiments

We experimented with our system to detect intrusions that exploit the known vulnerabilities of privileged Unix programs, `rdist` and `fingerd`. Expected normal behaviors of `rdist` and `fingerd` have been formally specified in [7,8]. The program `rdist` is a Unix utility for maintaining identical copies of files over multiple hosts. It has the flaw that when updating a file it can access the `chown` and `chmod` system calls for symbolic links. This flaw has been exploited by attackers to set the *setuid* bit of a system shell. (The normal behavior specification of `rdist` has many checkpoints, one of which requires that it can change the permission mode and the ownership of *only* the files that it created.) Consequently the attack described above can be detected by asking IDS if `rdist` has changed the permission mode or the ownership of a file that it did not create.

The second vulnerability we studied, `fingerd`,

reads finger requests using *gets* library call, which does not specify a maximum buffer length. An attacker may send a long request message to the finger daemon that overwrites the run-time stack with its own code. When the call to *gets* returns, it executes the attacker's code. (One of the normal behavior specifications of `fingerd` is that it can execute *only* the `finger` program.) So an attacker's attempt to execute her own code can be detected by asking our IDS if `fingerd` is trying to execute a file that is not the *finger* program.

The following example shows how our system would respond to pseudo-English sentences inquiring a trace about hypothetical `rdist` and `fingerd` attacks.

```
% rdist chmod ed a file that it not create ed?
'Not as far as I know'.

% fingerd execute ed a program that be s not "/usr/ucb/finger"?
['/tmp/hackersprgrm'].
```

4. Conclusion

In this paper we described our efforts to build a system that facilitates the interaction between intrusion detection systems and users through natural language interface.

As far as we know, there has been no previous research in this direction of direct intercourse between users and IDSs. It can be seen that our system is still rudimentary. We are currently extending the grammar of the language and implementing a wide variety of predicates, so that it will become more expressive in its ability to specify the normal behaviors of privileged programs.

Our future work will make more use of Prolog's deduction capability to detect intrusions. We are developing another version that runs constantly and detects intrusions in real time, rather than responding to queries, making our system able to learn normal behaviors and intrusion patterns, and we plan to extend our IDS from a single host-based system to a network-based system.

References

[1] J. Balasubramaniyan, J. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni, An architecture for intrusion detection using autonomous agents, *Proc. Computer Security Applications Conference*, 1998.

[2] N. Chomsky, *Minimalist inquiries: the framework*, MIT Press, 1998.

[3] M. Crosbie and G. Spafford, Defending a computer system using autonomous agents, *Technical Report No. 95-022*, Department of Computer Science, Purdue University, 1994.

[4] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff, A sense of self for UNIX processes, *Proc. IEEE Symp. on Security and Privacy*, pp. 120-128, 1996.

[5] S. Forrest, A. Perelson, L. Allen, and R. Cherukuri, Self-nonself discrimination in a computer, *Proc. IEEE Symp. on Security and Privacy*, 1994.

[6] H. Harkema, A recognizer for minimalist grammars, *Proc. 6th Int. Workshop on Parsing Technologies*, 2000.

[7] C. Ko, G. Fink, and K. Levitt, Automated detection of vulnerabilities in privileged programs by execution monitoring, *Proc. Computer Security Applications Conference*, 1994.

[8] C. Ko, M. Ruschitzka, K. Levitt, Execution monitoring of a security-critical programs in distributed systems: a specification-based approach, *Proc. IEEE Symp. on Security and Privacy*, 1997.

[9] E. Stabler, Derivational minimalism, in: C. Retor, ed., *Logical aspects of computational linguistics*, Lecture Notes in Computer Science 1328, pp. 68-95, Springer-Verlag, NY, 1997.

[10] E. Stabler and E. Keenan, Structural similarity, *Proc. Algebraic Methods in Language Processing*, University of Iowa, 2000.