A Knowledge-Based Approach to Network Security: Applying Cyc in the Domain of Network Risk Assessment

**Blake Shepard, Cynthia Matuszek, C. Bruce Fraser,
William Wechtenhiser, David Crabbe, Zelal Güngördü, John Jantos,
Todd Hughes, Larry Lefkowitz, Michael Witbrock, Doug Lenat, Erik Larson**

Cycorp, Inc., 3721 Executive Center Drive, Suite 100, Austin, TX 78731
{blake, cynthia, dcrabbe, zelal, jantos, larry, witbrock, lenat}@cyc.com
bfraser@gmail.com, william@wechtenhiser.com, thughes@atl.lmco.com, elarson@icc.utexas.edu

Abstract

CycSecure™ is a network risk assessment and network monitoring application that relies on knowledge-based artificial intelligence technologies to improve on traditional network vulnerability assessment. CycSecure integrates public reports of software faults from online databases, data gathered automatically from computers on a network and hand-ontologized information about computers and computer networks. This information is stored in the Cyc® knowledge base (KB) and reasoned about by the Cyc inference engine and planner to provide detailed analyses of the security (and vulnerability) of networks.

1 Introduction

In maintaining secure computer networks, system administrators face an increasingly time-consuming task. Much of the difficulty derives from the burden of information management: the amount of information required is enormous, much of it changes rapidly, and the relevant information can be difficult to identify. Existing tools are difficult to keep completely updated, and the format in which they provide information can be unwieldy. In this paper we describe CycSecure™, an emerging AI application in the domain of network security. This work endeavors to address some shortcomings of existing security software by exploiting strengths in the Cyc® technology [Lenat, D. B. and Guha, R. V. 1990], [Lenat, D. 1995]: a large knowledge base (KB), natural language input and output modules, a well-developed inference engine and an integrated planner.

CycSecure is a combination of information gathering and AI technologies, integrated into the much larger and more general Cyc system. CycSecure operates by scanning a computer network to aggregate information that describes the network's state. It then uses this information to build a formal representation or *model* of the network, based on Cyc's pre-existing ontology of networking, security, and computing concepts. The model incorporates information about what computers are on the network, what programs are installed or running on those computers, what privileges the running programs have, what users are logged into the computers, how the parameters of critical operating system files are set, *etc*. The formal model is stored directly in the KB, where it is accessible for inference and planning. This formal representation also allows users to interact directly with the model of the network using ontology editing tools, allowing testing of proposed changes or corrections to the network before implementation.

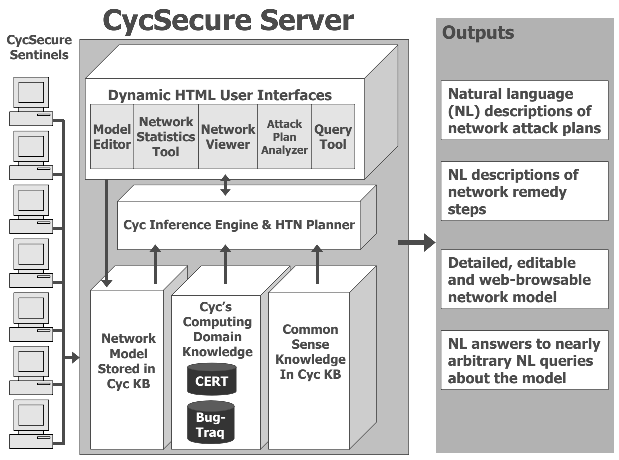

CycSecure Server
CycSecure Sentinels
Dynamic HTML User Interfaces
Model Editor
Network Statistics Tool
Network Viewer
Attack Plan Analyzer
Query Tool
Cyc Inference Engine & HTN Planner
Network Model Stored in Cyc KB
Cyc's Computing Domain Knowledge
CERT
Bug-Traq
Common Sense Knowledge In Cyc KB
Outputs
Natural language (NL) descriptions of network attack plans
NL descriptions of network remedy steps
Detailed, editable and web-browsable network model
NL answers to nearly arbitrary NL queries about the model

Figure 1. CycSecure Architecture

The planner has access to knowledge about software faults that have been reported on standard security tracking sites (primarily Bugtraq and CERT[1]). These faults are taxonomized according to a finely articulated ontology that includes knowledge about how to exploit each type of weakness. Users can direct the Cyc planner to assemble specific network attack plans that capitalize on one or more weaknesses in the modeled network. Cyc's inference engine reasons about plan structures to identify critical nodes in

[1] Bugtraq is available at: http://www.securityfocus.com. It is a public forum for reporting security faults. Fixes and patches, as well as exploits, are posted. The CERT website provides detailed assessments of software faults: http://www.cert.org.

attack plans to propose single-step remedies (*e.g.*, recommending downloading software patches to fix critical faults, or recommending specific alternative programs with the same functionality).

2 Application Description

2.1 Cyc Technology Overview

Cyc is composed of a large KB, an inference engine, a planner and natural language components.

- **The KB:** common-sense knowledge encoded in the CycL language. Approximately (as of March 2005) three million assertions (facts and rules) interrelating over a quarter of a million concepts.
- **The inference engine:** a resolution-based reasoning system, consisting of a general theorem-prover and a growing regiment of over eight hundred special-purpose modules. Each module is an implementation of a pattern of reasoning for a common type of problem.
- **The planner:** an extension of SHOP [Nau *et al.* 1999], which is an efficient hierarchical task network planner.
- **NL parsing and generation components:** a general lexicon, parsing, generation and discourse modules. NL generation underpins all CycSecure interfaces.

The common-sense knowledge in the KB consists, in part, of categorizations of objects and general truths about types of objects. Having this pre-existing framework simplifies encoding domain-specific knowledge, such as that required for CycSecure. CycSecure's knowledge focuses on topics such as computer programs and their functionalities, computer network vulnerabilities and network attack plan actions. The representations of these classes was facilitated by the existing representations of more "common-sense" concepts: types of information-bearing things, such as prose text, emails, computer viruses, applications and operating systems; concepts of information flow and information transfer; and even the basic concept of a computer, as a temporal, tangible thing. It was also useful to draw on the pre-existing representations of human agents as having motives, drives and behavior patterns, many of which are relevant to the way networks are used and thus affect security.

Given the broad range of existing ontological distinctions in the KB, it was clear how to incorporate the requisite new classes of security domain knowledge. A team of just 3-4 trained knowledge enterers was able to add all relevant knowledge to the Cyc KB in under 18 month's time. This rapidity was facilitated by the ability to forgo spending time stating relevant general facts, such as the fact that an electronic device that is turned on will stay turned on until its power supply is interrupted.

2.2 CycSecure Architecture Overview

The architecture of the CycSecure system is centered on a CycSecure server. The server in turn rests on a Cyc image, which includes the components described above, as well as a set of interfaces that allow users to interact with network models. The interfaces include an editor, a network statistics reporter, a viewer (for browsing information about different parts of the network) and an interface to the planner. The CycSecure server is designed to run on a dedicated machine. It handles all user interaction and KB interaction tasks.

Each client machine on the network runs a daemon (or *Sentinel*) that gathers information about the software, hardware and status of the machine it is running on. The server polls the Sentinels, gathers network information from them, and then represents that information in the KB.

2.3 CycSecure Software Components

CycSecure Sentinels and Server

The Sentinels are small software daemons that run on each machine on the target network. They are designed to consume very few system resources; such remote agents can be installed on even slow machines on a network without incurring a serious performance cost [Humphries, J. W. and Carver, C. A. Jr. 2000]. Each Sentinel is dormant until polled by the server. When polled, a Sentinel gathers information about the machine it is running on and relays it to the server. Sentinels are extremely platform dependent, but they share a common XML protocol understood by the server; as a result, they can be written for multiple platforms, potentially including dedicated firewall hardware or smart hubs.[2] The server translates the information returned by the Sentinels into CycL and adds it to the KB.

This approach is potentially vulnerable to an attack on the Sentinels themselves, or to spoofing of their communication. The latter concern can be addressed by encrypting the communication stream. However, since each Sentinel is sending unique information about the machine on which it is installed, the former concern is very real. Installing each Sentinel with a communications wrapper that contains a rolling key helps somewhat, but is still subject to some kinds of attacks and increases the complexity of upgrading the Sentinels over time.

CycSecure's architecture is not dependent on using Sentinels. It could also utilize third-party daemons that may have already solved some of these problems.

CycSecure User Interfaces

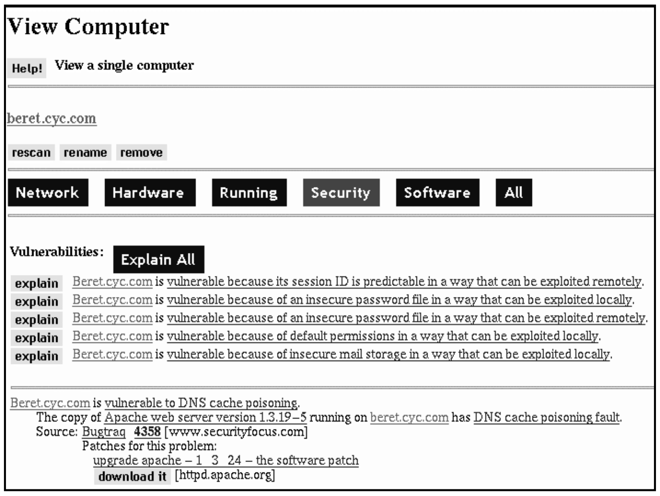
Users access the CycSecure server via several interfaces: the Attack Planner, the Model Editor, the Network Viewer, the Query Tool and the Network Statistics tool. Each of these tools is presented to the user in a web-browsing framework: the CycSecure server dynamically generates HTML and JavaScript in response to user interactions.

[2] To date, Sentinels have been developed for Windows 95/98/NT/2000/XP, Linux RedHat and Solaris.

The *Attack Planner* enables the user to state plan goals, launch the planner and view the attack plans generated. For example, a user can state the goal “An external user with no initial access gains administrator/root access to target.mynetwork.net.” The user then examines the plans returned, using the browser to get more detail about any interesting or unclear steps or components. (We discuss the Attack Planner in more detail in section 3.2, below.)

The *Model Editor* lets users edit the automatically built network model (*e.g.,* to add or remove machines or running programs from the model). CycSecure performs analysis against the model, so this tool lets users evaluate results of changes before deploying them to the actual network.

The *Network Viewer* allows users to examine and drill into the network model. For example, a user can view a computer and see: hardware specifications for that computer; programs running on it; information about a program, which may include links leading to known faults; information about exploits which can then be followed to a definition of buffer overflow exploits in general, *etc*.



Beret.cyc.com is vulnerable because its session ID is predictable in a way that can be exploited remotely.
Beret.cyc.com is vulnerable because of an insecure password file in a way that can be exploited locally.
Beret.cyc.com is vulnerable because of an insecure password file in a way that can be exploited remotely.
Beret.cyc.com is vulnerable because of default permissions in a way that can be exploited locally.
Beret.cyc.com is vulnerable because of insecure mail storage in a way that can be exploited locally.

Beret.cyc.com is vulnerable to DNS cache poisoning.
The copy of Apache web server version 1.3.19–5 running on beret.cyc.com has DNS cache poisoning fault.
Source: Bugtraq 4358 [www.securityfocus.com]
Patches for this problem:
upgrade apache – 1 3 24 – the software patch
download it [httpd.apache.org]

Figure 2. The Network Viewer Tool
In this screenshot, the tool displays NL descriptions of deduced vulnerabilities of a computer in the model; an explanation for one deduction is expanded into the view.

The *Query Tool* allows users to construct queries over predefined elements of the represented network and background knowledge such as installed programs, running programs, operating systems and IP addresses. The query is passed to the inference engine, which returns lists of results.

The *Network Statistics* tool provides a synopsis of the network state. It lists the number of machines on the network, the number of vulnerabilities on the machines, *etc.* This view of the network’s security state is useful for observing general patterns, but it does not indicate how any of the vulnerabilities may be exploited to compromise network resources. The Attack Planner imparts that more sophisticated capacity (see section 3.2, below).

3 The Ontology and The Planner

3.1 Ontology

The general knowledge already represented in the KB facilitates deep reasoning about security: it includes knowledge of communication protocols, computers, networks, firewall rules, corporate policies, employee/employer relationships and date and time functions, all of which are relevant to interesting questions pertaining to security.

Cyc’s KB underpins all of CycSecure’s capabilities. Some of the most pertinent portions are the ontology of program types and their functionalities and the ontology of software faults and their corresponding vulnerabilities.

There are currently 12,409 computer programs represented in the Cyc KB. At the top of this hierarchy stands the collection `ComputerProgram-CW`,[3] which is partitioned into `ApplicationProgram` and `OperatingSystem`. Important facetings of `ComputerProgram-CW` are `ComputerProgramTypeByPlatform` and `ComputerProgramTypeByFunction`. The latter faceting has 389 instances and includes collections such as `Screensaver`, `EmailClientProgram`, `NameServerProgram` and `LoginProgram`. This functionality-based faceting helps CycSecure present information in a structured (and therefore easily comprehensible) manner. *E.g*., users can select a view of computer that shows a list of its installed programs organized into a hierarchy of program types. Without a principled organization this list would be too verbose to be useful.

The hierarchy of program types is also used in vulnerability assessment and planning, since the same kind of software fault can lead to different vulnerabilities, depending on what kind of program has the fault. Security faults in software (as well as parameter misconfigurations) underpin vulnerabilities of physical systems; an email program with a buffer overflow fault yields a vulnerability that is functionally distinct from that of a printer driver with a buffer overflow fault. The planner takes these distinctions into account when generating attack plans, which are multi-step ways of capitalizing on vulnerabilities to achieve goals.

CycSecure’s ontologies of faults and vulnerabilities are novel because they are integrated into a much larger general ontology and are closely tied to planner rules. In the latter sense, the existing work most similar to CycSecure’s fault and vulnerability ontologies is that of Undercoffer and Pinkston [2002]. Their ontology of attacks is “target-centric,” emphasizing the means, consequences and locations of attacks as attack-type discriminators. CycSecure’s ontology of faults and vulnerabilities is likewise target-centric, in the sense that it has been built with the intention of supporting the generation of attack plans. An important

[3] The “`CW`” in the name of this collection indicates it is a specialization of `ConceptualWork`, the collection of deliberately created things that lack a location in space but have a beginning in time and an associated abstract information structure.

distinction between CycSecure's ontology and that of Undercoffer and Pinkston is that the first is intended to support vulnerability analysis, whereas the second offers an ontology of attack types in support of intrusion detection.

Currently 354 classes of software fault are represented in the Cyc KB. At the top of the relevant multiple-inheritance hierarchy is the general collection, `ProgramFault`, which has eighteen direct specializations such as the collection of denial of service faults, the collection of readable mail faults and the collection of input validation faults. Many faults in the hierarchy are direct specializations of multiple other faults. For example, `MountRequestForwardingFault` is a direct specialization of both `InsecureFilesystemFault` and `CommandFilteringBypassFault`.[4] If a mount request is for a filesystem that is mountable, a valid filehandle will be returned. This filehandle in turn can be used to access a filesystem that would otherwise not be accessible to a remote user. Because an unauthorized mount request should have been blocked, `MountRequestForwardingFault` is a specialization of `CommandFilteringBypassFault`, which is the collection of faults had by programs that fail to reject certain requests. A `MountRequestForwardingFault` can be exploited to gain unauthorized access to a file system, so it is a specialization of `InsecureFilesystemFault`, the collection of faults had by programs that can be made to yield a copy of a filesystem to an unauthorized user.

There are 683 classes of vulnerability represented in the KB. The most general is `ComputerVulnerability`, which has 28 direct specializations, including the collection of vulnerabilities to bounce attacks and the collection of vulnerabilities due to insecure login systems. A significant faceting of the specializations of `ComputerVulnerability` distinguishes remotely exploitable vulnerabilities from locally exploitable ones. To generate a complete plan in which a hypothetical attacker exploits a merely locally exploitable vulnerability, the planner must incorporate prior steps that grant the attacker local access.

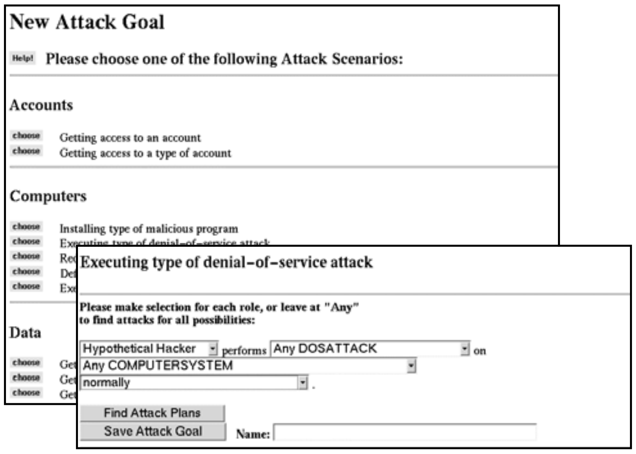
3.2 Planner

Cyc's planner is a variant of SHOP, an efficient hierarchical task network planner [Nau *et al.* 1999]. The planning domain is a representation of actions that could be performed using a computer system, especially those that could be performed by malicious hackers or disgruntled employees. Actions are represented as CycL formulæ with instances of `ActionPredicate` as their operators. For example,

```
(runProgramOnAs
  NetworkUser033 Workstation010
   (SoftwareVersionFn Mozilla 1.4)
   (UserAccountFn LocalWindowsLAN "33"))
```

means that `NetworkUser033` is running Mozilla 1.4, on workstation 10, with the privileges of the account with the unique identifier "33," on the local Windows network.

`ActionPredicate` is partitioned into `ComplexActionPredicate` and `SimpleActionPredicate`. Complex actions are tasks that can be decomposed into sequences composed of other complex actions or simple actions. Complex actions have preconditions and decompositions, and simple actions have preconditions and effects.



New Attack Goal

Please choose one of the following Attack Scenarios:

Accounts

Getting access to an account
Getting access to a type of account

Computers

Installing type of malicious program
Executing type of denial-of-service attack

Data

Executing type of denial-of-service attack

Please make selection for each role, or leave at "Any" to find attacks for all possibilities:

Hypothetical Hacker performs Any DOSATTACK on Any COMPUTERSYSTEM normally.

Name:

Figure 3. Selecting an Attack Plan

Users initiate planning by giving the planner a complex action sentence as a goal (Figure 3). For example, the goal `(doGetLoginInfoForAccount Hacker ?ACCT)` would cause the planner to find all sequences of actions that result in `Hacker` acquiring login information for any account. The predicate `doGetLoginInfoForAccount` itself decomposes into a sequence of actions, one or more of which may also be complex. Ultimately a plan will be returned only if, when each action is fully decomposed, the resultant linear sequence of simple actions is such that the preconditions of its components can be satisfied in the order of the sequence.

Here is an example CycL planner rule:

```
(preconditionForMethod
 (and (isa ?A LoginAccount)
      (privilegedAccountForSystem ?PRV ?S)
      (accountForSystem ?A ?S))
 (methodForAction
  (doGetLoginInfoForAccount ?AG ?A)
  (actionSequence
   (TheList (doGetCodeToExecuteAsOn ?AG
             HackersPacketSniffer ?PRV ?S)
            (waitForOccurrenceOfEventOfType
             ?AG (LogInToAccountOnFn ?A ?S))
            (sniffAccountLoginInformation
             ?AG ?A ?S)))))[5]
```

This rule says that one way to acquire login information (*i.e.*, the username and password) for a privileged account

[4] A program has a `MountRequestForwardingFault` if an unauthorized remote user can send a mount request to it that will then be forwarded to the mount daemon.

[5] This sort of planner rule is similar in intent to the procedural rules discussed by Shrobe [2002]. Like Shrobe, we aim to make the rulebase as general as possible.

(*e.g.*, a root or administrator account) on a computational system is to run a packet sniffer on the system, wait for someone to log into a privileged account, and then sniff the login information. The first action in this sequence is `doGetCodeToExecuteAsOn`, which is itself a complex action; a variety of additional planner rules state how to accomplish it under various circumstances. The second and third actions are simple actions (a waiting and a sniffing); there is no multi-step way that Cyc knows of to accomplish these acts. However, they do have effects that Cyc knows about. For every simple action represented in the planning domain, the effects of that action are also represented. For example, Cyc knows that sniffing login information implies knowing that information:

```
(effectOfAction-Props
  (sniffAccountLoginInformation
    ?AGENT ?ACCOUNT ?SYSTEM)
  (knowsLoginInformationForAccount
    ?AGENT ?ACCOUNT))
```

There are dozens of rules in the KB that state decompositions or effects of actions relevant to computer security. Using these, the planner will typically find a variety of plans, each of which exploits one or more vulnerabilities in the network. When run against a model of an in-house test network, the planner generated some plans that exploited multiple vulnerabilities and which our in-house security expert found plausible and surprising (see Figure 4).

We modified the SHOP algorithm for CycSecure in a number of ways. First, we retooled precondition matching to use the KB as the primary knowledge source. This change allows the planner to make full use of the Cyc inference engine. The second change was driven by the homogeneity of the network model. Networks are frequently populated with nearly identical machines running nearly identical software; it would not be unusual to see a network of a hundred computers, each running four distinct programs that have faults of the same class. If there is one successful attack plan on this network that exploits faults of that class, four hundred uninterestingly distinct plans result. To avoid a combinatorial explosion, we implemented *multi-binding* of variables. When the inference engine returns the bindings for a precondition, the planner retains them in groups by variable. This simulates the exploration of each branch of the search tree, done in parallel, with some reasonable overhead at each node. This approach to precondition binding ensures that the user is not overwhelmed with nearly identical plans.

Storing inference supports for each plan also enables an extra phase of analysis. Often the existence of possible attack plans is not interesting in itself; rather, system administrators care about determining what the dangerous vulnerabilities of the system are and prioritizing them in order of urgency. By analyzing plans' inference supports, CycSecure can answer the question, "What is the minimal set of conditions to address to block this set of plans?"

Plan generation takes on the order of minutes (or, in the case of very complex plans, a few hours). The time taken by the planner correlates with the number of plans returned, but given the complexity of the search space, the exact correlation is not readily predictable.[6]

Step 1. Hypothetical Hacker sends an email to ws.testnet.com, inviting its user to a particular website.
Step 2. Hypothetical Hacker constructs a data string designed to overflow the memory buffer for Real Player Version 7.0.
Step 3. Hypothetical Hacker sends the malicious data string to Real Player Version 7.0 running on ws.testnet.com.
Step 4. Hypothetical Hacker overflows the memory buffer for Real Player Version 7.0 running on ws.testnet.com.
Step 5. Hypothetical Hacker installs a sniffer program on ws.testnet.com.
Step 6. Hypothetical Hacker waits for the user of workstation.testnet.com to enter the account password.
Step 7. Hypothetical Hacker uses sniffer program to sniff the login information for a user account on the TestNet LAN on ws.testnet.com.
Step 8. Hypothetical Hacker uses remote hacking computer to send a valid username and password to ws.testnet.com.
Step 9. Hypothetical Hacker uses remote hacking computer to login to ws.testnet.com.
Step 10. Hypothetical Hacker downloads the MSIEXEC exploit program onto ws.testnet.com.
Step 11. Hypothetical Hacker runs MSIEXEC to compromise ws.testnet.com's operating system, Windows NT.
Step 12. Hypothetical Hacker gets access to an account with SYSTEM privileges on ws.testnet.com.
Step 13. Hypothetical Hacker can now read or write any file on ws.testnet.com.

Figure 4. Sample planner output, generated (in NL) by CycSecure running against a test network

4 Validation Field Trial

For a period of six months, members of the US STRATCOM Computer Emergency Response Team (STRATCERT) tested CycSecure. Feedback received during this process helped refine the capabilities of the tool. Although the data gathered during initial deployments was primarily qualitative, the results were quite positive. One interesting aspect was the ability to monitor compliance with IAVAs (Information Assurance Vulnerability Alert) issued by the Defense Department. (A sample IAVA might require that no workstation running Windows 9x run any version of RealPlayer earlier than 7.0.) Estimates of potential time spent on IAVA compliance monitoring (without CycSecure) are in the vicinity of hundreds of staff-hours per alert. To address this issue, CycSecure's querying capability was modified to allow users to define and save queries that correspond to specific compliance goals. For example, an IAVA can be converted into a query that finds machines in the model that are not in compliance with it. This capability

[6] Scaling to large networks can be addressed by adding computing hardware (since the planning process is highly amenable to parallelization). In addition, the use of dynamic multi-bindings enables planning times to remain relatively independent of network size.

addresses part of the compliance issue by providing a rapid detector for non-compliance. In future trials, more quantitative data will be collected on the system's speed, usability and accuracy.

5 Future Work

5.1 Planner improvements

CycSecure has proven to be a successful application of AI planning technology in a real-world domain. Logical next steps include applying CycSecure to larger networks and increasing the breadth and depth of our planning domain. We expect its knowledge-intensive planning approach to scale up to even the largest enterprise-level networks.

5.2 Software fault representation issues

Representing software fault reports in the KB is substantially faster than coding scripts to exploit them[7], but it still requires substantial time from a trained ontologist (about 10 to 20 minutes per report) because the task presents the usual ontology-building challenges encountered when building a very large knowledge base (*i.e.*, ensuring consistency, preventing redundancy, *etc.*). Cyc's underlying structure and knowledge entry tools are designed to make that task simpler, but it still requires expertise.

Although knowledge enterers' time need be spent only once, and can be shared across installations of CycSecure, a mechanism for automatically retrieving information from reports – either by parsing the natural language reports currently prevalent, or by taking advantage of a more structured reporting system – would be a logical improvement in the process. Current work is focusing on improving natural language retrieval mechanisms and providing better tools for ontologizing semi-structured information.

5.3 Configuration representation

The information currently gathered by CycSecure from networks focuses on data required for adequate vulnerability assessment, based primarily on knowledge of software faults, installed software and high-level descriptors of running computers. One extension should be the inclusion of representations of configuration options chosen for common applications, particularly those that offer communication capabilities, such as web servers and browsers, mail servers and clients, and remote login programs. Such programs, if misconfigured, can affect the security of a computer or a network as a whole.

[7] Most vulnerability assessment software operates by running canned attack scripts against a network. The Nessus scanner (http://www.nessus.com/) is one of the best open-source examples of this kind of software. Due to the difficulty of writing exploit modules for this kind of system, unfortunately, Nessus currently covers only a fraction of the faults reported on Bugtraq.

6 Conclusion

By providing a semantic underpinning for information collected from a computer network combined with a rich existing corpus of more general information, CycSecure gives its users a deep analysis of the state of their network, particularly by demonstrating feasible attack plans that could be launched against it. Widespread application of CycSecure and adoption of its suggested remedies should provide better system security and substantial timesavings. Test users have found the technology to be valuable, and its extensibility will enable it to provide more powerful analyses as the breadth of relevant information represented grows. There are a number of interesting directions this technology could evolve. Some work has been done on modeling user access policies and on counterplanning, and interesting problems present themselves with respect to scalability and network management. However, the current application has the potential to present immediate benefits to users.

References

Aslam, T., Krsul, I. and Spafford, E. H. 1996. A Taxonomy of Security Vulnerabilities. *Proceedings of the 19th National Information Systems Security Conference*, 551-560.

Du, W. and Mathur, A. P. 1998. Categorization of Software Errors that led to Security Breaches. *Proceedings of 21st National Information Systems Security Conference,* 392-407.

Fithen. W. L., Hernan, S.V., O'Rourke, P. F. and Shinberg, D. A. 2004. Formal modeling of vulnerability. *Bell Labs Technical Journal* 8(4):173-186.

Gorodetski, V. and Kotenko, I. 2002. Attacks Against Computer Network: Formal Grammar-Based Framework and Simulation Tool. *Lecture Notes in Computer Science Volume 2516 / 2002*, Springer-Verlag.

Howard, J. 1997. *An Analysis of Security Incidents on the Internet*. Ph.D. thesis, Carnegie Mellon University.

Humphries, J. W. and Carver, C. A. Jr. 2000. Secure Mobile Agents for Network Vulnerability Scanning. *Proceedings of the 2000 IEEE Workshop on Information Assurance and Security*, USMA, West Point, NY.

Lenat, D. B. and Guha, R. V. 1990. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley.

Lenat, D. 1995. Steps to Sharing Knowledge, in *Toward Very Large Knowledge Bases*, ed. N.J.I. Mars. IOS Press.

Lindqvist, U. and Jonsson, E. 1997. How to systematically classify computer security intrusions. *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 154-163.

Nau, D., Cau, Y., Lotem, A. and Muños-Avila, H. 1999. SHOP: Simple Hierarchical Ordered Planner, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI99)*, ed., T. Dean, 968-973.

Shrobe, H. 2002. Computational Vulnerability Analysis for Information Survivability. *AI Magazine* 23(4):81-91.

Undercoffer, J. and Pinkston, J. 2002. Modeling Computer Attacks: A Target-Centric Ontology for Intrusion Detection. *Proceedings of the 2002 UMBC Center for Architectures for Data-Driven Information Processing Research Symposium.*