The Management and Security Expert (MASE)

Mark D. Miller, Stanley J. Barr, Coranth D. Gryphon, Jeff Keegan, Catherine A. Kniker, Patrick D. Krolak[1]

University of Massachusetts at Lowell
Center for Productivity Enhancement
One University Ave
Lowell, Massachusetts 01854

Abstract

Today's computing environments are increasingly complex: they often consist of large networks of computers that include multiple vendors and operating systems. The various systems and the communications between them must be kept running at their peak performance levels to meet the demands of the user community. They must also be kept safe from malevolent intruders and damaging viruses. The challenge facing today's systems manager is an enormous one. Unfortunately, the conventional tools provided by manufacturers provide only minimal assistance, leaving the system manager with the bulk of the work.

The Management And Security Expert (MASE) can help. MASE is a distributed expert system that monitors the operating systems and applications of a network. It is capable of gleaning the information provided by the different operating systems in order to optimize hardware and software performance; recognize potential hardware and/or software failure, and either repair the problem before it becomes an emergency, or notify the systems manager of the problem; and monitor applications and known security holes for indications of an intruder or virus. MASE can eradicate much of the guess work of system management.

Introduction

The Management And Security Expert is a distributed system capable of monitoring operating system resources and network statistics across an entire network. MASE will consist of customizable expert system modules running on each host in the network, along with the addition of expert systems that provide special functions such as network security, network performance, and network audit services.

---

[1]Mark D. Miller, Project Manager; Stanley J. Barr, Research Assistant; Coranth D. Gryphon, Research Assistant; Jeff Keegan, Research Assistant; Catherine A. Kniker, Research Assistant; Patrick D. Krolak, Center Director

MASE is implemented using PCLIPS, or Parallel CLIPS[1], which comprises a set of extensions to NASA's CLIPS language. PCLIPS was designed by the ULowell's Center for Productivity Enhancement (CPE) for rapid prototyping of distributed applications, and has since expanded to include dynamic construct creation, archival facilities, truth maintenance, and alarm-timing mechanisms for cycling execution of constructs, and timed expiration of facts and instances.

PCLIPS is set up in a Client/Server model. Under the current model, each PCLIPS client has it's own PServer. When one expert system communicates with another, it does so by calling the *send-message* function, which connects with the PServer, and transmits a *send* message, which includes the outgoing fact. The PServer then in turn sends a PFACT out to the server of the other PCLIPS client.

Host Level Management and Security

MASE utilizies a series of diagnostic operations, or functions, written to monitor and check various system parameters and resources. At this point in the development of MASE, we have developed four types of diagnostic operations: the *check, get, action*, and*display* operations.

The *check* operations are used for checking a system resource against either a system manager defined threshold, or a previously saved system state. As the system manager configures each expert system, s/he sets the frequency in which the *check* operations are run. (Ex. 30 seconds, 6 Hours, 1 Week, 1 Month) Examples of *check* operations:

check-disk-flood, check-boottime, check-sys-clock, check-rhosts

The *get* operations are used for retrieving system values and settings. They are used when either the system manager or the CLIPS expert system is trying to solve a problem. Examples of *get* operations:

get-load, get-boottime, get-users, etc.

The *action* operations are used for fixing situations. They can be initiated by the system manager, or by the expert system. Examples of the *action* operations:

action-move-file, action-disuser-user, action-compress-file, etc.

The *display* operations are used for displaying system information in a form other than facts. This is typically done using a graphic interface. This allows the system manager to remotely monitor a situation, and take appropriate action if necessary. Examples of the *display* operations:

display-cpu-usage, display-disk-usage, display-queue

Depending upon system configuration, these operations can be performed as run-time functions, or they can be run as spawned (standalone) processes. When these operations are performed by MASE, they generate facts, which are used to drive the controlling expert system. If a particular operation is run as a run-time callable function, then the *assert* function is used to

---

[1]"PCLIPS: Parallel Clips", Coranth Gryphon, Mark Miller, Second Annual Clips Users Conference

add the fact to the PCLIPS fact base. When the operation is run as a spawned function, it connects to the PServer of the parent PCLIPS client, and sends a local fact (LFACT). The PServer then passes the fact along to the client.

The facts that are generated by the diagnostic operations have the following form:

(RESULT <result> <type> <func-name> <tag> 'descriptive message')

The **result** field has one of five values:

success - This fact type is asserted when the operation completes successfully, and does not find any problems. (Used in Check operations)

warning - This fact is asserted when the operation finds a problem with a system resource. (Used in Check operations)

error - This fact type is asserted when the operation cannot successfully perform the task it has been assigned. (Ex. Missing configuration file - Used in all four types of operations)

outcome - This fact type is asserted when the operation has accomplished some task. (Used in Action operations)

information - This fact type is asserted when the operation has obtained some information. (Used in Get operations)

The **type** field has one of several values:

disk, memory, cd-rom, op-disk, network, process, op-sys, user, files, security, peripheral, zones, server, misc, unknown, "some string" - reserved for later use.

The **func-name** field is the name of the function that generated the fact.

The **tag** field is used for distinguishing between facts created by different calls to the same function. For instance, if the function is called twice, with two separate sets of parameters, the tag field allows PCLIPS to distinguish between the two calls.

The **"descriptive message"** is the rest of the fact that has the actual result information in it. For example, a warning fact generated by the check-disk-flood operation would look like this:

(RESULT warning disk check-disk-flood DEVICE "/dev/rza1" at SETTING 98 exceeding THRESH 95)

In the **"descriptive message"**, the capitalized words (DEVICE, SETTING, THRESH) are used as tags, signaling PCLIPS that the next field is information that should be extracted.

Configuring the host management expert system

Since the system manager configures each expert system, the reactions of the expert system can be tailored to the specific needs of the manager. For example, take the situation where the expert system is monitoring the amount of space available on a disk. Disks, and their associated storage space are critical resources to a running system. When a disk hits 100% capacity, users can no longer write to it. This is especially critical to users who are in editors, or are running applications that generate information that needs to be stored. During normal operational hours, the situation may not become critical, because the system manager would be able respond. However, at 3 a.m., there may not be any personnel around who can fix the problem. Because of this possibility, the following scenario typifies how MASE might be

configured to solve a disk flooding problem on a Unix system. (See Fig. 1 for the user interface that the system manager would use)

Initially, the function *check-disk-flood* would be set up with a frequency of once every 30 minutes. The threshold passed to the *check* function for checking disk space on a partition might be 90%. When this threshold is exceeded, the expert system notifies the system manager of the situation, via a fact to the system manager's expert system controlled user interface. The expert system then modifies the threshold to 95%, and the frequency to 15 minutes.. If the threshold is exceeded again, the system manager is appraised of the new circumstances. Since the disk is approaching a completely flooded state, the expert system checks the time, which is 8:37 A.M. This time falls outside the range of normal system manager hours, so the expert system checks to see if the system manager is logged on and active (This can be done by querying other expert systems on the network). If the system manager is not available, the expert system takes an active role in resolving the situation. It performs the following operations:

1) Notify all the users that there is a problem, and ask them to take action themselves. (Clean up directories, compress files, etc.)
1) Check the temp directories on the disk (*check-tmp-dirs)*. Purge any files older than 2 days that are in these directories.
2) Check for core files on the disk (*check-for-cores)* that are more than 1 week old. If any are found, compress them.
3) Check for any tar files on the disk (*check-for-tars)*, and compress them.
4) Get a new reading of the disk space (*get-disk-space).*
5) If the percentage is below 90%, the problem has been corrected, so reset the values in the *check-disk-flood* function.
6) If the percentage is between 90% and 95%, set the parameters lower on the problem solving functions, such as using 1 day as a parameter for *check-tmp-dirs.*
7) If the percentage is higher than 95%, drastic actions may be required

a) Delete core files (*check-for-cores)* which generates a fact that contains the pathname of each core file found. These facts will in turn cause the rule that calls the function *action-delete-file* to fire.
b) Check the file system for a disk with more space.(*check-fs-usage*) If one is found, it may become necessary to move files over to it. Create a storage directory (*action-create-directory*) and move files over to it, starting with tar files. Any time a file is moved, the owner of the file must be notified of the move. (*action-notify-user*)

Notifying users becomes another issue. If mail is used, this may compound the flooding disk problem. So, the expert system might check to see if the user is logged on. If so, a message is written to the user's screen. If the user is not available, one option might be to insert a line in the .login of the user. Another option might be to keep the name of the user in the factbase, and start a check function that periodically checks to see if the user is logged in. When the user does log in, write to the screen of the user, and remove the fact from the fact base.

Currently, *action-notify-user* is set up to only to write to the screen of the user. However, that function could realistically be converted to a mini-expert system, which solves the problem of notifying the user without using mail.

The critical element of this system is that it is completely system manager configurable at startup, and modified dynamically at run-time. These operations are policy decisions, which are made on a per-site basis.

Configuring the host security expert system

Since security is vitally important on some installations, and not all that important on others, the system manager can set the level of security for the MASE system, on each node. That way it is possible to have a higher level of security on critical systems. The trade off is higher security requires higher resource utilization for MASE (more CPU time, more memory, etc.). At this point, we have three security stages defined. Stage 1 is the highest, Stage 2 is a medium level of security, and Stage 3 is a low level of security. This will change in the future, as we model national guidelines for computer security.[12] Here are some examples of the security levels for a BSD Unix system.

Stage 1 (High) - Inter-MASE messages would all be encrypted.
No device, including the console, would be set to *secure*. That way, no user could log onto the system directly as root. Another account would have to be used first, followed by an *su*.

Stage 2 (Medium) - Some inter-MASE messages would be encrypted, such as registration messages.
Only the console could be set to *secure*, but the system manager could override that.

Stage 3 (Low) - No inter-MASE messages would be encrypted. (Fastest system)
The system manager can set any device to *secure*.

There are two areas of interest in security at the host level. The first area is *Hole Detection*. MASE can be programmed to check for holes in the operating system, such as a world writeable password file, or world readable /dev/kmem. Since MASE is expert system driven, it is easy to add the ability to check for a new hole, should a new one arise. If a hole is detected, MASE will, depending upon it's configuration, either close the hole, or simply notify the manager. It checks for weaknesses that have been created by users, intentionally, or not. For instance, on some versions of the Unix operating system, there is a file that can be created by a user, called the .rhost file. In it, the user can list what machines and accounts can log onto the user's account, without the use of a password. In this file, the user is allowed to use a wildcard, which creates a tremendous security hole. When a wildcard is specified in the username field, it creates a hole that allows any user on the specified machine to log in on the account, without specifying a password.
Examples of some of the hole detection check functions are:

check-os-files-protection - This checks a list of os files that need specific protection settings.
check-suid-files - This checks for files with the suid bit set.
check-passwd-file - This checks for exploitable accounts (no passwords, unauthorized root uid

---

[1] "*A Guide to Understanding Configuration Management in Trusted Systems*", National Computer Security Center, March 1988

[2] "*A Guide to Understanding Trusted Facility Management*", National Computer Security Center, October 1989

owners, etc.)

The second area of security at the host level is *intruder detection*, which is in the preliminary stages of development. This requires MASE to monitor the areas of the operating system that are prone to attack, such as password guessing. The following scenario shows how MASE could be configured to handle a password guessing attack.

There is an attempt at logging onto the system. A valid username is provided, but the password does not match. MASE detects the failure. If a second attempt is made, MASE then attempts to determine where the login request is coming from. This information could be obtained through *intelligent* utilities at the operating system level, or network packet analysis. If three attempts fail (manager definable amount), MASE could then start putting up barriers.

1) If the attempt is being made from outside the manager's domain, it is possible, using an intelligent network router, to disable incoming traffic from the offending system, or domain. This is fairly drastic, but if the security of the system is deemed important enough by the system manager, then it is a useful response.

2) Temporarily disuser the account, until the real owner of the account identifies him or herself.

One of the advantages of MASE reacting to an intruder threat, instead of simply notifying the system manager, is that valuable time could be lost waiting for the system manager to react.

Configuring the network management expert system

At the network level , MASE can use both passive and active actions to diagnose network related problems. In order for the system manager to understand the layout of the network, it is important to generate a topological map of the network, which can be displayed for the system manager. The topological map is created through the use of several different types of network functions.

1) Pinging: This is an active function that uses network software to determine the existence of an active node on the network. Using a list of expected nodes on the network, (/etc/hosts on Unix systems) the pinging function will actively poll the various nodes on the network. If they are alive, they will respond back to the ping. This gives MASE a initial picture of what nodes are active on the network.

2) Routing Tracing: This is a second active function that allows MASE to trace what routing paths would be used between two specified machines. This enables MASE to get a clearer picture of the network topology. Using this technique, which uses UDP packets, MASE can determine where gateways and routers are located, address-wise.

3) Ethernet Monitoring: Turning a Unix machine into promiscuous mode allows MASE to monitor packets traversing the network. Using a filtering program, statistical information based upon traffic patterns can be gathered. This can be used for simple informational purposes, and it can also be used to diagnosing network problems, such as saturated subnets, malfunctioning ethernet cards, run-away daemons, etc.

4) SNMP Capabilities: The Simple Network Management Protocol (SNMP) has become the de facto standard for a network management protocol. At this point, most hardware vendors have either delivered, or promised to deliver SNMP support with their network hardware. SNMP support will allow MASE to gather network information from network machines, without having

to run MASE on each of the machines. This becomes even more important for network routers. Routers are not multi-function machines. They are designed with one purpose in mind, which is routing of network traffic. Instead of trying to get a MASE agent running on different routers, MASE will use the SNMP protocol to get the same type of network information. This way, the MASE developers do not have to port MASE to the various types of network hardware, such as routers.

Eventually, MASE will progress to the point where the expert systems will act as a network management adviser. Based upon statistical information gathered by MASE, it might suggest how a network could be reorganized, based upon traffic flow, and resource utilization. For instance, it may suggest, as a result of traffic analysis, that the insertion of a bridge at a certain point in the network will reduce the load on each side of the bridge by 40%. Or, it may suggest how to distribute a file system better, so that a particular node which is being over-utilized will share some of its load with a second node.

Configuring the network security expert system

Computer security on a network can be analyzed at various levels. The lowest level is the machine level. It could be argued that the security of a network is only as strong as the security on the weakest node on the network. This is an overstatement, but not as irrelavent as one might think.

At the network level, MASE will use it's packet monitoring capability for several security checks. An example of one follows:

In every ethernet packet, the hardware address of the ethernet card which transmits the packet is included. If it is an Internet Protocol (IP) packet, the IP address of the machine is included also. Since MASE will be monitoring network traffic, it can compile a list of all the physical ethernet addresses, and their associated IP addresses. One method of attacking a computer network is to disguise one machine so it looks like another. This is done by changing the internal networking information on the system. This type of attack would be detected by MASE, however. Changing the IP address of a machine is fairly simple, given the right privileges. However, changing the physical ethernet address of the ethernet card is not. When the disguised machine starts transmitting packets using the false IP address, MASE will detect the anomaly (conflicting physical addresses) and report the problem.

Since MASE is a distributed system, the various nodes can supply information that can lead to detection of other types of intruder attacks. For instance, if one node reports that an account was accessed improperly ten times (invalid passwords), then a problem may exist. However, if a second machine reports shortly thereafter that it has an account that is being tested (multiple password guesses), and the originating user is using the flagged account on the machine that intially reported the password attack. This should raise the possibility of an intruder attack to a much higher level.

User Assistance

One of the problems that users face on a network of computers is knowledge of, and access to, network resources. MASE will assist users in this regard. It will keep track of the various network resources, such as printers, disk drives, tape drives, application software, development software, etc. This will be maintained in a database, which the user will then be able to query against. Example:

A user has a dvi formatted file, that s/he wants to print out. A query would be made to MASE, asking what machine has a dvi to postscript utility, a postscript printer, and access in some way (User Account, Guest Account, etc).

System Manager's Interface

First, it is important that the system manager is shown what the network looks like, topologically. Second, the system manager needs to be able to access information quickly on each of the nodes. Third, the system manager needs to be able to configure MASE to meet the requirements of the particular network. This includes thresholds, corrective actions, notification levels, and frequency settings.

The MASE user interface is designed to handle each of these items. The display is set up hierarchically, enabling the manager to move up and down throughout the network, displaying items such as subnets, routers, individual nodes, node resources, and finally, the expert systems that are being executed. Each item is represented iconically, allowing the user to select an item, in order to obtain more information about the item. For instance, if the item selected by the user is a subnet, then all the nodes on the selected subnet are displayed.

One of the features of this interface is a graph-building capability. This allows the system manager to design and implement his/her own expert systems, for solving problems specific to his/her own network.

Finally, one of the problems of user interfaces for complex systems, is information overload. This is overcome in MASE, by using the manager's expert system to control the interface. Messages coming in from across the network are prioritized. For instance, a message alerting the system manager that a printer is down does not carry the same importance that "an intruder detected" message does. The controlling expert system will use the priority scheme to determine which information should be displayed before other information.

Conclusions

MASE is an on-going project that is continuously evolving. It is being funded through a RICIS contract for NASA Johson Space Center. The first phase of the project was to develop an Alpha version of MASE.. This included basic diagnostic function development, preliminary user interface development, and expert system development.

MASE goes a step further than other network packages, by providing complex rule-generation systems that allow system managers to create their own custom automated networking experts. The ability to combine existing commands and expert systems into larger, more intelligent systems is a more powerful and dynamic solution than today's rigid, stiff alternatives.