1 Introduction

Perimeter Control has conventionally been the primary focus of attention in enterprise network security. By regulating access in and out of the network through firewalls and proxy servers, users and system administrators were always provided with a sense of isolation and implied security that unfortunately often failed to materialize.

Besides the numerous and well documented strategies for circumventing perimetral defenses, the emergent risk of the insider threat essentially put to rest any remaining hopes that perimeter defense alone (even if correctly implemented) could provide adequate protection to enterprise networks.

As consequence, a change in focus began to emerge, shifting the point of attention from the perimeter of the network to the host itself. If each and every host in the network is adequately protected, not only the risks of a compromise due to a breach in the perimeter are greatly minimized but so is the footprint of the damage. That is, a successful attack in one host in the network will not necessarily imply that the remaining hosts are immediately vulnerable. The concept is simple and effective: redundant layers of defense to mitigate the likelihood of success and the consequences of an attack.

The problem, however, lies in the details. Maintaining the security of individual hosts in networks with heterogeneous systems, policies, and capabilities quickly became a major task. System administrators were required to maintain detailed descriptions of each host and their related vulnerabilities, which changed frequently. Furthermore, they were also expected to be able to quickly identify and correct issues as necessary.

To support the new paradigm system administrators and security practitioners quickly coordinated to organize communities to monitor and report vulnerabilities on several operating systems and applications. Operating system and
application vendors also joined the effort, building update agents to facilitate the process of patch distribution and updates. These powerful communities have provided (and continue to provide) and invaluable service to improve not only network security in general but also to raise awareness and attention to the issue.

Successful attacks, however, continued to occur with increasing frequency and effectiveness. Although the number of novel attack strategies continues to decline, old techniques still exploiting previously known and well reported vulnerabilities are still in the rise. A telling example of the problem is the MSBlast worm that took place in 2003.

The MSBlast worm was officially reported on August 11th, 2003 (CA-2003-20). The following day, the CERT Coordination Center (www.cert.org) identified as many as 1.4 million unique Internet addresses appearing to be sources of infection for the worm (Lemos, 2003). Leaving aside all the details about the worm, the exploited vulnerability had been identified and reported well before the incident. On July 17th, roughly one month before MSBlast hit, Microsoft and The United States Computer Emergency Readiness Team (US-CERT) released a vulnerability report (VU#568148) and security advisory (CA-2003-16) reporting the issue and recommending procedures to patch the operating system.

Cases like this are certainly not rare. Studies have shown that even thirty days after the discovery and announcement of a vulnerability only half of the affected systems were patched (Lemos, 2003). In contrast, according to Symantec’s September 2005 Internet Security Threat Report (Symantec, 2005), attackers can create exploits for publicly reported vulnerabilities within six days of the announcement.

Our focus of concern in this work is not necessarily the end user directly connected to the Internet through now pervasive high speed links. Instead, we turn our attention to enterprise networks that usually maintains trained administrative personnel for system and network administration and security. These types of networks are the most critical to the cyber-infrastructure in general, as they constitute the most business, financial and medical institutions, military and government networks and others. Successful attacks in such networks are prone to cause chain effects, with larger impacts and losses.

In enterprise networks, the system administrator in charge is faced with a set of challenges that are often overlooked. As illustrated in Figure 1, system administrator and security personnel have to be aware of several pieces of (often incomplete) information.

- First they must know well their network, which include each system, application and configurations.
- Another piece of information is the, often undocumented, Corporate Knowledge and Culture. Such knowledge includes the costumes and practices of the corporation and varies significantly from company to company. For instance, a military of government network might establish a complete different set of rules, autonomy and responsibility to system administrator than an Engineering company providing design services would.
- Aware of the network and their responsibilities, system administrators must also track information about current vulnerabilities and corrective procedures. Such information has to be mapped to the network (constrained, if you will by the culture of the enterprise) and only then the recommended corrective action is identified.
- The execution of corrective actions for a given vulnerability is yet another challenge. Administrator must have timely access to each vulnerable system, the knowledge and the necessary tools for the task.

![Figure 1. The challenges of system administration.](image)

1 CERT® advisory number, available online at www.cert.org
must ensure that, on each particular system, changes made will not cause unwanted side effects or new vulnerabilities.

The process is essentially continuous, that is, while taking corrective actions for one vulnerability (which might actually span one or more days of work), new vulnerability announcements arrive and system configuration change. The problem is aggravated by the fact that, in most enterprises, users are granted at least some administrative capabilities to their system and unintended changes in configuration, software updates or re-installations often revert previously corrected problems, accounting the persistent reoccurrence of old vulnerabilities (Lemos, 2003).

The ultimate consequence is essentially what it observed today in several enterprises. System administrators learn to prioritize their efforts and compromise. Vulnerabilities are only checked against a small set of hosts in the network (often servers) and administrative control to such machines is highly constrained. Such practices, however, do not suffice when user machines are compromised either while in the network or not, and overall productivity is affected.

In this paper we describe MAST, a mobile agent-based security tool designed to provide a comprehensive solution to the problem. The goal is to ensure two core capabilities to system administrators: a) to provide support to the cognitive tasks involved with the problem, and b) to provide the necessary support for the technical and administrative tasks of patching or correcting to address the vulnerability.

The paper will first present an overview of the approach proposed in MAST, followed by a description of its architecture and core components. We will then present and discuss the experimental results obtained during a controlled test with a group of system administrator, and conclude the discussion with a brief summary and analysis.

2 The MAST Approach

MAST combines two core capabilities that we argue as required for an effective and comprehensive security tool. It combines a) a cognitive support system based on shared knowledge models that are readily accessible to user, and b) a network and system management tool to provide a novel mechanism to ensure that administrative tasks are carried with the appropriate coverage and maintained into effect.

The tool integrates and builds upon two key technologies (Fuggetta et al., 1998): mobile software agents and concept maps (Novak and Godwin, 1984). Used together, these technologies allow for the construction of a single system that satisfies both the technical support and cognitive support requirements.

Concept maps are graphical knowledge representations that have been extensively used in several aspects of education, knowledge elicitation and sharing. In MAST, concept maps are used as tools to explicitly define and share information about network system as well corporate policies and procedures. Several studies (which will be discussed later in the paper) show that concept maps are very effective for such tasks.

Mobile agents are independent units that can be dispatched over a network connection to execute on remote hosts (Fuggetta et al., 1998). Once dispatched, mobile agents operate autonomously without constant supervision and the need for a persistent connection to the original host (unlike remote computation or remote invocation). The agents have the ability to move around the network based on an itinerary or a set of locally checked conditions given by the administrator.

One of the most powerful capabilities offered by mobile agents is the ability to push new code to a system and execute the code in a disconnected manner. This allows a server to dispatch agents to remote hosts which then execute remotely and independently. The ability of the agents to carry new code to the remote system allows on-the-fly development and deployment of custom monitoring and management tools.

The notion of delegation allows agents to act as representatives of the user who launched them, inheriting his credentials for access control and accountability. In general, agents can be imbued with different levels of autonomy that might allow, for instance, a possible change in plans under different network conditions.

In MAST, mobile agents are used as the enforcers of system administrator’s tasks. They are self contained software units that can move between machines and locally execute administrative tasks under policies defined at the agent system (infrastructure) level. They are analogous to actual security guards roaming a building at night and repeatedly checking every door and window.

Security guards have local knowledge of the building, including policies and common threats; they individually perform simple tasks, but jointly they ensure the overall physical safety of the building. Similarly in MAST, software agents are designed (or configured) by system administrators with embedded knowledge about systems and applications that are specific to their domain and that must be checked. Instead of just applying a necessary patch or update to the core OS, these agents are capable to monitor and correct arbitrarily complex conditions.

Furthermore, security agents can operate disconnected from the network and can quickly suspend and resume execution to support operations in laptops and other portable devices. When moved outside the network (during a trip, for instance), such devices will continue to operate under local monitoring of the previously deployed security guards, ensuring continuous compliance with security policies.
The SK abstracts the underlying system and application-dependent details of each host. It provides a standard cross-platform Java API, as well as an extended cross-platform APIs that supports a number of common security related tasks such as user account management, process handling and high level access to the file-system. The SK is based on a hardened and streamlined version of the NOMADS agent system (Suri et al., 2000a). NOMADS provides two implementations: Oasis and Spring. Oasis incorporates a custom Java-compatible Virtual Machine (named Aroma) whereas Spring is a pure Java implementation.

The Aroma VM is a clean room implementation of the Java Virtual Machine developed at the Institute for Human and Machine Cognition. It provides to critical capabilities for agent systems: a) fine grained resource utilization and accounting control on a per agent and per process basis, and b) support to strong mobile, that is, transparently moving an agent between hosts in mid-execution, or between two consecutive bytecode instructions (Suri et al., 2000b).

The Spring implementation is fully interoperable with Oasis; it does not support strong mobility, although it may be modified to provide fine grained resource utilization and accounting control on a per agent and per process basis, and b) support to strong mobile, that is, transparently moving an agent between hosts in mid-execution, or between two consecutive bytecode instructions (Suri et al., 2000b).

The SK abstracts the underlying system and application-dependent details of each host. It provides a standard cross-platform Java API, as well as an extended cross-platform APIs that supports a number of common security related tasks such as user account management, process handling and high level access to the file-system.

The SK is based on a hardened and streamlined version of the NOMADS agent system (Suri et al., 2000a). NOMADS provides two implementations: Oasis and Spring. Oasis incorporates a custom Java-compatible Virtual Machine (named Aroma) whereas Spring is a pure Java implementation.

The Aroma VM is a clean room implementation of the Java Virtual Machine developed at the Institute for Human and Machine Cognition. It provides to critical capabilities for agent systems: a) fine grained resource utilization and accounting control on a per agent and per process basis, and b) support to strong mobile, that is, transparently moving an agent between hosts in mid-execution, or between two consecutive bytecode instructions (Suri et al., 2000b).

The Spring implementation is fully interoperable with Oasis; it does not support strong mobility, although it may be modified to provide fine grained resource utilization and accounting control on a per agent and per process basis, and b) support to strong mobile, that is, transparently moving an agent between hosts in mid-execution, or between two consecutive bytecode instructions (Suri et al., 2000b).

Figure 2. Combining knowledge models with roaming security agents.

Differently than previously proposed mobile agent-based intrusion detection systems (Jansen, 2002; Karygiannis, 1998), MAST is a distributed security checking tool designed to facilitate proactive patching and maintenance of security on hosts and networks. MAST is essentially human-centric in the sense that security agents are constrained by human instructions and policies. Agent autonomy in MAST is used only to allow the reporting of abnormal or unexpected conditions.

3 MAST Architecture and Components

MAST was designed to be a flexible system. The idea is to allow system administrators to customize the tool to their network and corporate culture. The first structural design of the tool was proposed by Carvalho et al. (2003), and greatly evolved to the current design discussed in this paper. Figure 3 illustrates the core components in MAST, namely, the security kernel, MAST Server, MAST Console, MAST Agents and the Knowledge Models. The Security Kernel (or MAST) is installed on every host in the network and supports some core functionalities like agent mobility, monitoring, intra-agent communication and accounting. The other components are either self-contained or rely on some for security kernel capabilities.

3.1 The Security Kernel

The Security Kernel (SK) is the foundational component of the system. SKs are installed on all network hosts and form the security substrate, handling the execution, the mobile and communication between software agents.
Communications between security kernels and the MAST server is authenticated and encrypted via TLS (Dierks and Allen, 1999). When first installed, there is an exchange of certificates between two entities using an off-channel key. Once authenticated, SKs can interact with the MAST server, and provide secure communication services for agents and other components.

3.2 The MAST Server

The MAST Server is a distributed, abstract entity rather than a centralized host. It is composed of a set of mobile agents that do not need to reside in the same location. The server agents run on SKs, this means that non-server agents can migrate and execute on server hosts. The System Administrator can dictate which systems on his/her network may be eligible to host server agents (trusted machines), and how many agents can execute concurrently on each host.

Server agents will automatically replicate and migrate to different machines (if so specified by policies) to improve fault tolerance and load-balance. If one of the active server hosts fails, the remaining replicas of the server agents residing on that host will replicate again and move to alternative trusted machines. SLP (Guttman et al., 1999) is used as the primary mechanism for service lookup.

There are six basic service agents currently provided with MAST, an Agent Lifecycle Service, an Agent Communication Proxy Service, an Authentication Service, a Vulnerabilities Management Service, and a Logging Service.

The Agent Lifecycle Service provides launching, tracking, recovery, and termination of agents in the system. For security reasons, core agents are stored in a restricted access codebase location and can only be launched by the Agent Lifecycle Service.

The Agent Communication Proxy Service provides the communication infrastructure between agents and the MAST Console. The Authentication Service has direct access to the server keystore, and authenticates SKs, agents, as well as the human operators of the MAST system.

MAST also includes an integrated vulnerabilities database that maintains a comprehensive set of parsed and pre-processed vulnerability and security advisory
As illustrated in Figure 4, a fundamental difference between these two types of agents is that for each task initiated by the user, there will be a single instance of a sequential agent residing in the network, while in the case of parallel agents there will be multiple independent instances of the same agent residing in different locations at the same time. This represents an important factor in determining how the user interacts with each agent, and will be further discussed in item 3.4.

3.4 The MAST Console

The MAST Console is a graphical interface to the system. It is used by system administrator to monitor each host (security kernel), the MAST Server (or server agents) and also every running agent in the network. The console also incorporates logging facilities, account management and multiple agent-interaction interface methods. A screenshot of the MAST Console is shown in Figure 5.

The console relies on authentication and secure-communication mechanisms that are similar to those provided by the SK. After user authentication at the console, both the underlying kernel and any agents launched from that console will inherit the credentials of the user. Based on user credentials, the console allows different levels of access to agents and systems.

This architecture ensures that agents initiated by a specific user will inherit and maintain appropriate privileges even after the console shuts down. The agent, tagged with user credentials, will persist in the network and (if configured to do so) will wait for the user to log back into an administrative console to report its activities. The console is only an interface; it provides a view of the
system but is completely independent of the agents that it launches, monitors or controls.

The console also includes facilities for scheduling the launching of agents and an embedded editor to modify and create new agents. Scheduling allows agents to be run at a later time on arbitrary machines, or to run periodically or continuously, to ensure that changes are not reverted back (or overridden) by other users or components.

### 3.4.1 Modes of Human-Agent Interaction

Once dispatched, interactive agents can remotely create a graphical interface to communicate with the user. Examples of when this interface could be used are, for instance, when the agent’s operation requires user input, or when local conditions at the host (e.g. a file lock) may prevent the agent from carrying out its task.

In the case of the sequential agent, every time a query to the user is required, the administrator will be prompted by a single graphical interface. For parallel agents, on the other hand, each instance of the agent might display its own interface to the administrator.

In order to efficiently interact with a large group of agents, MAST provides an IRC-like interface called Group Manager (GM) (Carvalho et al. 2006). A Group Manager interface is created for each agent dispatched in parallel to a set of machines. In this case, when multiple instances (or clones) of the same agent are created and launched to a number of hosts, the agents will form a common functional group, with similar tasks and behaviors at each machine.

The Group Manager then builds an IRC-like graphical interface that allows the user to simultaneously exchange text messages with any subset of agents in the group. The interface is particularly useful for parallel agents. An example of the Group Manager interface is shown in Figure 6.

Text messages can be easily broadcasted to all selected agents and each response block is tagged by the agent prefix in the message board. Messages are parsed and handled asynchronously by each copy of the agent in their respective target environments.

Figure 7 shows a snapshot of a group of parallel agents using both interface methods simultaneously. The individual agent dialog boxes are shown on the left side of the console (for only a small number of agents), while the Group Manager interface for the same agents executing the same task is shown on the right side of the console. Although the number of messages is the same, the Group Manager interface facilitates the grouping of events and allows simultaneous instructions to be sent via text messages.

The protocol utilized to communicate with agents via text message is also configurable. When an
agent is created, a system administrator can specify a set of text commands associated with a brief description and mapped to specific methods implemented by the agent. That information can be queried later (via text messages) and used as an online help for interacting with the agent.

Figure 6. The Group Manager Interface.

Figure 7. Comparing the Group Manager interface with individual agent dialog boxes.
3.5 The Knowledge Models

Concept maps (Novak and Gowin, 1984) have been widely used to organize and represent knowledge in several application domains. Concept maps are basically characterized by a graphical representation of a set of concepts and their relationships, providing a strong and concise description of the specific domain of knowledge. The concepts are hierarchically arranged and linked to created propositions, which are usually referred to as semantic units or units of meaning. Due to its lack of formalism and intuitive nature, Concept maps are often used in knowledge elicitation sessions directly by the experts, greatly facilitating the process of elicitation (Novak and Cañas, 2005). Figure 8 shows an example of a concept map.

In the context of MAST, concept maps are used to organize and facilitate access to information about many different aspects in network and host security. MAST relies on cmapTools\(^2\), to facilitate the online manipulation of concept maps. cmapTools is a powerful concept mapping software (Cañas et al., 2004) that has been customized in this project to support direct communications with the MAST Server and the launching of security agents in MAST.

The concept maps in MAST can also be utilized in training sessions for novices. New system administrators can review basic concepts in network security through the knowledge models (Figure 9), and drill down to very corporate-specific information as needed.

As a security or system administration tool, MAST does not depend on the knowledge models to operate. Every task can be directly performed via the MAST Console, independently of the knowledge models. The concept maps are available from the console as a support for the administrators, who can also opt to browse the knowledge base and interact with the agents directly from the knowledge models representing the context in question.

This approach allows some degree of separation between the access to the security tool and the knowledge models that can also be used by cmapTools for other purposes such as training and education. For instance, when agents are launched from an authenticated console, the cmapTools application can be used as a monitoring and managing tool as well as a knowledge browser and editor. When used independently, it works as knowledge management tool, with direct access to all the maps and models in the system.

4 The Security Model

MAST’s architecture involves the interaction of multiple network components in addition to mobile code. These capabilities, although fundamental for the flexibility

---

\(^2\) The CmapTools software package is freely available for non-profit use at (http://cmap.ihmc.us)
of the system, do raise several security concerns (Farmer et al., 1996; Vigna, 2004) that must be addressed at the level of the framework, or SK substrate.

4.1 Intra-kernel Trusted Relationship

Communications and code mobility are always handled by the security kernels. In MAST, there is no direct interaction between two components or agents residing atop different kernels. Every message, command or mobility request passes through the underlying SKs.

All network communication between the SKs in the MAST system is encrypted by TLS and authenticated by means of mutual authentication utilizing PKI. During the initial MAST setup phase, the public key of each SK is signed by the same organizational Root Certificate Authority. The Authentication Service’s Public Key will be distributed to each SK during this setup phase. These data elements, verified during the TLS handshake, will then serve to provide the mutual authentication between all SK’s and the Authentication Service. Given that the system may be deployed on networks that rely upon DHCP for IP dissemination, and/or utilize NAT Gateways, we found that we cannot rely upon IP Addresses or Domain Names, items traditionally used within X.509 certificates (Housley et al., 1999) for validation. Also, given that our Server Modules are mobile agents, we cannot depend upon them being at the same IP for any two subsequent transactions. Additionally, incoming connections are only allowed to the SK from the local subnet, or an address range explicitly specified in the local security policy. A unique identifier automatically created for each security kernel is utilized instead.

4.2 Agent Mobility

Agent tampering by a host execution environment is one of the major security issues raised by mobile agents. If an agent is traveling from the MAST server to host A and then to host B, there is no guarantee that host A will not tamper with the agent before sending the agent on to host B. This is particularly true if host A has been compromised and the SK on host A has been altered. This type of mobility is commonly referred to as multi-hop mobility.

While several approaches have been proposed to handle multi-hop mobility in a secure manner, none of them currently provide a satisfactory solution. Therefore, the
default security policy in MAST limits agent mobility to single-hop. That is, an agent may migrate only between the SK running the Agent Lifecycle Service and another SK. In certain specialized applications, if the administrators need multi-hop mobility and have other means of guaranteeing the integrity of the SKs, they may change the security policy to allow multi-hop mobility.

In order to avoid the multi-hop security problem, sequential agents are always required to move back to the MAST Server for integrity checking before proceeding to another kernel. Sequential agents are those that move through a specified sequence of hosts in a serial fashion. This extra intermediate hop is transparent to the user.

4.3 Launching Agents Securely

An action taken in the Knowledge Model (KM) or the MAST Console will typically generate an agent launch request. In the case of the Knowledge Model, the console operator will have identified the need to launch agent(s) to a particular host(s), and requested this launch via the pertinent Concept Map. This request will be transmitted to the co-resident MAST Console, which would verify that it came from an authorized KM browser. If the launch request originated directly from the MAST Console, this step is omitted.

The console would then transmit this launch request to the Agent Lifecycle Service, who will initiate and dispatch the requested security agent, with the provided arguments (if any).

4.4 User and Agent Authentication

The Authentication Service maintains the security levels and the association between the operators and their assigned trust levels. There is one administrative or root authority level whose holder is responsible for adding each human operator into the system. The ability to browse the KM is intended to be a basic ability and is available to all users who have access to the system(s) that are authorized by local security policy to run the MAST Console. Further authentication is required by means of an assigned security token for higher level functionality, including launching agents and authorizing agents to make changes directly on client and server hosts.

4.5 MAST Console Security

The MAST Console is the interface between the users and the system. The authentication between console and MAST server is exactly the same as the Security kernels and the MAST server. The Console has to be initialized and bootstrapped during the installation process.

There are two levels of authentication. The console application first authenticates with the server and builds a persistent secure channel. User authentication is based on username/password pairs provided by the user and with the server through the previously established secure channel. Based on pre-defined server policies the user will be granted the appropriate level of access to the system.

The Knowledge Model Browser, a standalone application, will have been invoked from the Console context, and the authentication established by the Operator currently on the Console will have been transferred, by means of a dynamically generated key, to the KM Browser during this step. This key is then utilized by the KM Browser to authenticate itself back to the Console when requesting agent launches.

5 Experimental Evaluation

The experimental evaluation of the framework consisted on having groups of system administrators perform a number of security-related administrative tasks (with and without the support of the MAST framework) on a controlled set of Windows and Linux systems.

While not using MAST, participants were allowed to use any of the commonly available system management tools such as ssh, netstat, remote desktop, etc. While using MAST, the participants had the option to use a set of basic agents provided with the framework to handle simple system management commands such as ps, netstat, etc. Alternatively, participants could also write their own agents for specific tasks.

Experiment participants (19 in total) were placed in the role of a system administrator responsible for conducting a pre-defined set of security related tasks in a small network of 11 hosts. All hosts are connected to a LAN (either via 10BaseT or 802.11) and configured either with a static or dynamic IP address (DHCP). The participants were given a map of the network and direct access to all the machines with administrative permissions. Each participant was given the following tasks:

- Identify the machines in which a specific user account is present and disable that account.
- Determine which machine has the most free disk space.
- Identify the machines in which a specific P2P file sharing application is running and terminate the process.
· Enable Windows Update (and schedule automatic updates) on each Windows host.
· Determine which machines are infected by a specified self-propagating worm and remove it (the only way to remove the worm was to first isolate the affected machine(s) from the network).

The goal in each run was to perform as many tasks as possible within a fixed time period. The experiment was performed in two parts. First, each participant was given access only to the conventional system management tools. They were then asked to perform the proposed tasks in any order within a 45 minutes time limit.

During the second part of the experiment, the participant had access to the MAST framework to perform the same set of tasks, within the same time limit. The group of participants was divided such that approximately half of the group started the experiment without MAST, while the other half started with MAST. All participants did both sessions, with and without MAST.

At the end of the experiment, each participant was independently interviewed and given a short survey with questions about different aspects of the MAST tool, and about the overall experience.

All participants had system administration background and, with very few exceptions, no formal training in programming. Agents written by the participants during the experiment (while in the MAST session) were, in most cases, modifications for already existing agents provided as part of the framework.

On average, all participants of the experiment were able to accomplish a higher number of tasks using MAST than they did with conventional tools. The normalized distributions of each case are shown in Figure 10, with their associated averages and standard deviations.

The level of confidence provided by MAST is significantly higher. That is essentially due to the fact that the actual execution of a task is brief and feedback from agents is immediate. Participants spent some time preparing and coding the agent but when dispatched, they received an immediate description (including logs) of everything that happened in all systems. That seems to provide a lot more confidence in the execution and completion of the task than handling host by hosts and annotating (or simply remembering) the results on each case.

When asked about the utility of the Group Manager interface to interact with groups of agents in medium to large scale networks, approximately 72% of the participants recognized the interface to be very useful, while 17% felt that the interface was useful, but not essential for performing large scale tasks.

There was very little negative feedback aside from the difficulty reported on writing security agents. That is, in a sense understandable and justifiable by the lack or previous programming proficiency in Java in our sample. We believe, however, that this problem can be mitigated with better agent design interfaces including, for instance, functional building block graphical agent design tools or other programming languages like Perl and Python.
6 Conclusions

Enterprise network security continues to be a complex and critical issue in cyber-security. It is our claim that an effective solution to the problem must address both the issues related to the cognitive load imposed to system administrators during the decision making process, and the issues associated with the effective execution of arbitrary corrective tasks.

In this paper, we present MAST, a mobile-agent based security tool designed specifically to address these issues in a flexible and effective way. We have chosen to design MAST on top of two core technologies: a) concept maps, for knowledge representation and sharing, and b) mobile agents for task distribution and execution.

We have described and discussed some preliminary experimental tests conducted to identify the effectiveness of the tool. Although recognizing the small sample and the simplicity of the tasks used in our tests, we are encouraged by the results and confident of the benefits provided by the tool.

MAST is a research project, and its main objective is to foster better understanding and science advance. As such, we have largely utilized the tool in several real-life networks and we’re confident to its applicability and capabilities. MAST is publicly available for non-profit use. It includes the source code of the base security agents (distributed with MAST) and the knowledge models. The knowledge models are also open to the community for voluntary further development and discussion.

7 Acknowledgements

The MAST project was sponsored by the National Science Foundation (NSF) under the Strategic Technologies for the Internet program, award number 0230927. Special acknowledgments to Chris Eagle, who directly participated in the MAST project and greatly contributed with the design, implementation and tests of the framework.

---

3 MAST is freely available at http://mast.ihmc.us, with documentation and contact information.
References


