An Integrated Software Immune System: A framework for Automated Network Management, System Health, and Security

Michael Gilfix
Tufts University
Medford, Massachusetts, 02155
Email: mgilfi01@tufts.edu

Abstract

Maintaining the integrity of large-scale networks is becoming an increasingly daunting task as networks expand at an unprecedented rate. The majority of present network monitoring and maintenance tools require a substantial investment in human resources to sift through vast quantities of information, to detect problems, and manually resolve them. Computer Immunology is the solution to ever-increasing network maintenance overhead. This paper seeks to define computer normality through a policy-driven process and provide a framework for conserving network health. The goal is to automate network management as much as possible. This methodology combines the benefits of two existing systems as well as incorporates other crucial elements to provide an integrated, flexible, versatile system.

1. Unwieldy Networks and Today's Computer Immune Software

Mark Burgess recently introduced the powerful idea of Computer Immunology [1]. Here at Tufts, like many other large-scale academic environments, such a solution has long been needed to handle escalating network demands. An immune system must be capable of using convergent processes to enforce complex site policies to be truly effective. It should also be capable of self-preservation to ensure that policy is enforced during critical periods. Also, network administrators need a language for defining health policies.

At first glance, CFEngine appeared to provide our much-needed solution: assuring system health of all kinds, in terms of configuration, security, and dynamic behavior. Unfortunately, CFEngine fails to assure total system health for several fundamental reasons, most notably its inability to fully interact with a system's OS at all levels and its limited facilities for defining complex rule sets. Therefore, PIKT[2] was explored as a possibility. But PIKT, too, had its own fundamental weaknesses and focused on too narrow a domain of system health.

A true computer immune system requires a synthesis of CFEngine and PIKT concepts while integrating other vital elements. Components of the system must have the ability to communicate with each other. The immune system must be able to interact with every aspect of the computing environment. It requires an expressive and inclusive language for defining rules. Moreover, a facility for storing and deploying a vast array of rules, the lifeblood of the immune system, is necessary.

2. Basic Immune Logic – Using Normality

The recognition of "abnormal" behavior and its neutralization drives the immune engine. The basic policy behind the engine is best described by what is considered intolerable behavior. The policy should provide for the following:

- Programs misbehaving or using unacceptable level of resources should be terminated.
- The user environment should not be allowed to clutter to avoid potential problems
- Should machines misbehave, they must be probed for errors and their configuration checked. Problems should be fixed whenever possible (through a CFEngine-like mechanism) and, as a last resort, system administrators alerted with detailed information if necessary.

This policy implicitly creates a division between local (i.e. first & second criteria) and state (third criteria) normality. Local normality concentrates on local machine resources and local processes whereas state normality involves machine configuration. An immune component (PIKT-like) should police local resources. When it encounters troubles it cannot resolve, it should call on an external mechanism (CFEngine-like) to examine machine configuration and perform the necessary changes. This should minimize loss of service and ensure the problem is addressed with immediate priority.

Burgess suggested that a computer immune system would need to recognize new patterns in program code and act accordingly. Such an analysis is too resource costly. The key is to isolate the actions of the user or the program in its environment. Thus, one can make an educated guess. If a program is misbehaving, it's most likely for the following reasons:

1) The program has bugs
2) The program is intentionally misbehaving
3) There's a configuration error

In the first case, we can only treat the symptoms, not cure the problem. In the university environment, buggy student programs crash constantly and many programs exhibit buggy behavior. So we need only treat such programs as if they were intentionally misbehaving. The cures for the second and third symptoms are within our grasp: we can kill a misbehaving program and tweak machine configuration.

Consequently these general policies lay a framework for the inner-workings of the immune system.

3. A Language for Health Rules and The Rule Database

A successful immune system would require a thorough description of its environment; the major work in assuring system health is to create the rules describing system illness. In an ideal world, one would know rules for everything. Moreover, this is not a one-person or one-institution process; everyone must become involved. This would prove impossible when using a tool controlled by a single text configuration file; a database maintainable by a large number of people is required.

The rule definitions for an immune system need to be both extremely expressive and flexible. Both CFEngine and PIKT's greatest weakness is the inability of their scripting languages to address a wide problem domain. By using a true rule-based programming language such as Prolog[3], rules will be able to describe extremely complex situations and address a large range of situations. By its very nature, Prolog also paves the way for the future incorporation of artificial intelligent interfaces.

The rules would then be stored in and downloaded from a central repository on the Internet using LDAP or a similar protocol. This will allow delegative administration of the rule sets by multiple parties, breaking the bottleneck of labor intensive gatekeeping that may well be the undoing of the current GNU project. A control machine or "brain" computer on each network would work in tandem with CFEngine and PIKT-like mechanisms to enforce and preserve system health. Using a control machine would give an administrator control over his network –such as which rules to enforce. (See Figure 1)

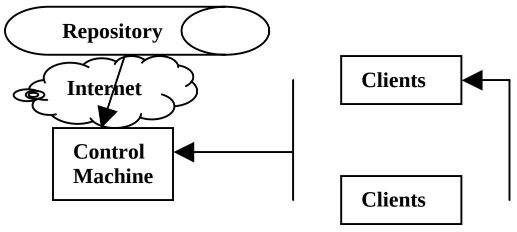

Figure 1 – A layout of the integrated immune system

The idea of the control machine could also be extended to analyze a wide variety of network information (such as SNMP queries), determine problems, code its own rules, and deploy them through out the network; such behavior dips into the realm of artificial intelligence.

4. Summarizing: The Integrated System

The truly powerful computer immune system is a policy-driven convergent process. The concept of normality provides the basics for such a policy. Combining a powerful, flexible rule-based language to describe system health with mechanisms that monitor and manipulate both machine configuration and resources provides an appropriate basis to keep machines and networks constantly functional.

References

[1] Mark Burgess, "Computer Immunology", Proc. LISA-XII, 1998

[2] PIKT. Created by Robert Osterlund. *http://pikt.uchicago.edu/pikt/index.html*

[3] Alva Couch, "It's Elementary Dear Watson: Applying Logic Programming to Convergent System Management Processes", Proc. LISA-XIII, 1999