An Architecture for Self-Protecting Autonomic Systems

GENE 401 - Special Directed Studies

Michael Jarrett
msjarret@uwaterloo.ca

August 29, 2003

Abstract

Autonomic computing is the concept of designing complex information technology environments with the ability to perform management tasks on their own without human support. These systems have been defined by their ability to self-configure, self-optimize, self-heal, and self-protect. Administrators would only be required to specify the high level requirements of the system, rather than the mundate configuration of individual componts, greatly reducing the human requirements for administration of large systems.

Self-protection refers to the ability of an autonomic computing system to secure itself against intrusions, and react to protect itself when it detects that an intruder has successfully circumvented the security policy on the system. While many of the individual elements that would be required for such abilities exist, very little research has been performed on how one would combine these elements to create an autonomic security system to meet this criteria of self-protection.

A structure for autonomic computing security systems is proposed by mapping responsibilities that such a system would have to fulfil onto a basic architecture for an autonomic computing system as described by IBM. This hierarchy is divided up into three layers. The highest layer deals with policy inputs from the administrators themselves, both as generic system-wide security policy, and as policies inherent in the requirements for each application to be run on the system. The middle layer translates these into applicable policies for collections of elements and coordinates the actions of individual autonomic elements at the bottom layer. In the bottom layer reside the autonomic elements that provide the services of the system, and are ultimately responsible for detecting and responding to intrusions.

Many components of such a system currently exist, and have been researched for many years, such as policy specification, intrusion detection systems, and rational agents. Other components, such as intrusion response and management technologies, are more recent research topics, and not often explored in the context of autonomic computing.

It is proposed that an environment be designed to allow for experimentation into autonomic security structures. This could be based on a future release of autonomic computing technology, on a new system created using existing distributed communication technologies, or a simulation for experimental purposes.

Once such an environment is available, research can be done into autonomic computing security structures and components. One interesting area is that of intrusion response, and more specifically that of a security through diversity approach; where the autonomic computing environment can respond to threats by replacing components with differing implementations.

Table of Contents

List of Figures

1 Introduction to Autonomic Computing

In October 2001, IBM published a manifesto describing their "Perspective on the State of Information Technology."[1] The document described what they viewed as a growing problem in the information technology world - that computer systems are becoming more and more powerful, and as applications were created to make use of this ever-increasing power, information technology environments were becoming more complex. This increase in complexity has resulted in computer systems taking more and more human resources to manage, with this demand for management growing in a way that will ultimately be unsustainable.

As such, they proposed a switch of focus to *autonomic computing*. The concept was initially inspired by the human autonomic nervous system, which handles critical tasks such as breathing, digestion, and one's heartbeat automatically without the need for a person to actively think about it. Similarly, autonomic computer systems would take care of day-to-day management without the need to involve human operators. The human operators would feed in high-level policies, which the system would configure itself to satisfy to the best of its ability.

IBM's manifesto lists eight key attributes of an autonomic computing system, describing how they envisioned what an autonomic computing system would consist of. Derived from this are four key properties of an autonomic system, which have been quoted by almost any entity with interest in autonomic systems. [2] [3]

- **Self-Configuring:** The system configures itself and its components to satisfy high-level objectives, and can change this configuration as needed.
- **Self-Optimizing:** Adapting to demand on the system, making maximum use of the resources available to it.
- **Self-Healing:** Handles unexpected failure in hardware and software, and can to repair itself (or bypass broken components) without interrupting the running system.
- **Self-Protecting:** The system can detect unauthorized intrusions and respond to these. The system actively attempts to secure itself against intrusion.

These four properties have become the standard way of defining an autonomic computing system. While some research still makes references to the biological analogy to the human

nervous system, most now rely on defining the technology instead as any system that can satisfy at least one of these properties.

Implied in each of these four properties (and the first on IBM's list of characteristics) is the aspect of self-knowledge. To manage themselves, these systems must become aware of their own abilities, as well as the abilities of those they interact with.

1.1 Key Corporate Players

While IBM is widely credited with both initiating and leading the push for autonomic computing, there are many other companies that have also spearheaded efforts into autonomic computing technology.

Each corporate player offers their own IT management software (Tivoli/System Management Server/Openview), plus other offerings, which in many cases determine how the initiatives differ. To date, no company has offered a complete operational autonomic computing infrastructure, instead choosing to integrate the concepts of autonomic computing into their existing product lines. This has led to a variety of largely incompatible incremental improvements in the products of each vendor, but does little in the creation of complete autonomic computing systems on larger scales.

1.1.1 International Business Machines

http://www.research.ibm.com/autonomic/

IBM lead the way with it's 'autonomic manifesto'[1], where it outlined what it viewed as a unsustainable information technology complexity problem, and proposed autonomic computing as its solution. It went so far as declaring that it would recenter its entire research division around the new notion.

While little has been done to form a standardized autonomic computing architecture, IBM has been quick to create software components demonstrating some elements of its autonomic computing push across a wide set of their products. Examples include a 'site analyzer' for their WebSphere e-business platform, which monitors and interprets website visitor traffic and usage, helping making e-business decisions, and also actively searches out and isolates database errors and contacts the administrator to apply a fix. Also, a variety of autonomic computing tools for their Tivoli system management platform have been announced, encompassing software distribution, storage, and centralized management. Recently they have

announced new technologies for 'adaptive forecasting', 'rapid reconfiguration', and 'on-line capacity planning', again with plans for incorporation in their WebSphere e-business platform.

IBM is perhaps alone in actually publicizing their vision of autonomic computing systems. One of the most relevant documents published is their description of an architecture of an autonomic system[4], where they outline the components of an autonomic system and the structure of an 'autonomic manager' in terms of 'control loops'. They are also one of the few to actually propose a generic structure of autonomic managers from a security perspective[5].

1.1.2 Microsoft

http://www.microsoft.com/management/default.mspx

While Microsoft tends to avoid the word *autonomic* itself, they have recently unveiled a large set of technologies concentrating on self-management under the moniker of the 'Dynamic Systems Initiative' (DSI)[6]. This initiative encompasses a variety of Windows-based technologies, including *Microsoft Operations Manager* (MOM), and *Microsoft Systems Management Server* (SMS), encompassed under a new XML-based *System Definition Model* (SDM). While light on details, Microsoft describes SDM as a way both for SDM-aware software to communicate with each other and with management software, as well as a standard way for management to represent system requirements and objectives. Promised is support for SDM under new releases of Microsoft's Visual Studio .NET development tool suite, which will allow application developers to ultimately develop applications that interact with MOM to self-optimize and self-heal.

Microsoft intends for the Windows operating system to have initial support for SDM in their 2004 *Longhorn* release of Windows with their entire management infrastructure scheduled to be finalized by the *Blackcomb* release, scheduled for 2006. Visual Studio .NET integration of SDM and related tools should be complete later this year, but very few details have been released thus far.

Microsoft has released *Automated Deployment Services* (ADS) and the Windows *Systems Resource Manager* (SRM) with Windows Server 2003, in the hopes of building support for its version of autonomic computing. ADS is a system to allow software to be automatically be deployed to a managed set of Windows servers, while SRM allows processor, memory, and other IT resources to be managed and distributed to particular tasks and applications. With the release of these tools, Microsoft hopes to improve its reputation in terms of system management to garner support for later released in the DSI.

Microsoft has announced the support of several partners for its initiative, including IBM, Fujitsu, and NEC. Many of these efforts involve support for the vendors' hardware platforms, though NEC has stated that it intends to incorporate the effort into it's own autonomic computing push.

1.1.3 Hewlett Packard

http://managementsoftware.hp.com/solutions/sol_0001.html

Under the name of the *Adaptive Management Platform*, HP is adding a wide selection of autonomic enhancements to its HP OpenView multi-platform management suite, as well as HP's Utility Data Center. A true advantage to these tools is their ability to integrate with many other applications including offerings from Microsoft and IBM. System-specific enhancements to HP servers as part of the Adaptive Management Platform are also being offered.

HP plans to make specifications and APIs for the HP Utility Data Center autonomic systems available later in the year.

1.1.4 Sun Microsystems

http://wwws.sun.com/software/learnabout/n1/

Sun's N1 platform, a catch-all name for most of their latest server platform, is looking to support the autonomic computing initiative as well.

> It is not Sun's intention (at least so far) to replace conventional Systems Management Software for monitoring such as CA's Unicenter, Tivoli, or BMC, but rather to sit beside it and send and receive information from such systems.[7]

Slightly different than the other offerings, Sun's N1 is looking to provide some of the lower-level virtualization that could be used in other management system. This could eventually lead to self-optimizing systems. However, recently Sun has been demonstrating self-configuring characteristics in several of their storage systems, so more things may yet be planned.

1.1.5 NEC

http://www.sw.nec.co.jp/innovation/it/valumo/english/

NEC is working on what they call a "platform concept" for VALUMO ("VALUe" + "MOre"). The key points highlighted are those of autonomy, virtualization, and distribution, and what little English language details are available describe a system much like what IBM describes as an autonomic computing system. Unfortunately, there is little in the term of English language details published on the VALUMO platform outside of NEC.

While not a product in itself, VALUMO serves as a concept that can be applied to their variety of products on both the hardware and software levels. Microsoft has indicated that it is collaborating with NEC to incorporate support for technologies based on their Dynamic Systems Initiative into the VALUMO platform.

1.1.6 Palladia Systems

http://www.palladia.net/

While still deep in 'stealth mode', this company has made bold promises to deliver what they consider to be the first autonomic computing infrastructure developed independent of an existing product line. Their "Angela" product will provide a base on which advanced autonomic applications could be developed.

Angela works on the basis of IBM's eight characteristics of an autonomic computing environment, plus adds the requirement that the system "must operate in an ecosystem". The environment in which the system operates must be considered as well as the system itself, according to Palladia.

Differing from many other companies, Palladia has placed a strong focus on self-protection. An entire architecture has been developed around determining whether a particular piece of data is in fact secure. Extra efforts have been made to add virus prevention capabilities. Even physical security will be taken into account in an attempt to secure systems.

Unfortunately, due to the lack of serious publications from the company, nothing can be stated for certain about the results of their efforts until their first product release.

1.2 Academic Work in Autonomic Computing

Many research groups have taken up the call to develop autonomic computing technologies. Most have chosen to focus on very small subsets of autonomic computing, or to related

technologies that could be used in an autonomic system. Some work independently, while others under the close watchful eye of one of the major corporate players.

While many research institutions have projects that can be related in some aspect to autonomic computing, only those that are heavily quoted as supporting the field are mentioned here.

1.2.1 University of California, Berkeley

One of the most-quoted research projects of the autonomic movement is the Berkeley/Stanford Recovery-Oriented Computing(ROC) Project [8]. This project is looking to develop methods to, rather than prevent software failures, to detect and recover from them quickly. Experiments at Stanford University to add 'recursive restartability' to the *Mercury* project [9] - a software system used to manage a satellite uplink base station - met with great success.

Though not directly related to autonomic computing, Berkeley's Computer Science department have several projects frequently quoted by those working on autonomic systems. One is the Telegraph project, which is an adaptive dataflow system, used for aggregating and analyzing data brought in over the network, and adapting dataflow to manage unpredictable network performance. Another project is the OceanStore project, which is a massively distributed persistent replicated filesystem. While neither of these projects are autonomic, some of the features of autonomic computing (self-optimizing and self-healing respectively) can be implemented on top of these platforms.

1.2.2 Columbia University

http://www.psl.cs.columbia.edu/dasada/

The DARPA-funded *Dynamic Assembly of Systems for Adaptability, Dependability, and Assurance* (DASADA) concentrates on creating autonomic systems and systems of systems running legacy software. Specifically the *Kinesthetics eXtreme* (KX) project provides an implementation of these designs based on a variety of XML communication technologies. The system defines a four-tiered model using interlinked sets of software 'probes', 'gauges', and 'effectors', linked over a series of standard communication busses.

The project has produced a prototype for KX, apparently available for download from their website, though initial requests to access this software were denied.

One appealing factor of this project is that it provides a number of communication technologies and underlying architecture on which autonomic computing concepts can be experimented with openly.

1.2.3 Carnegie Mellon

http://www.cmu.edu/

Carnegie Mellon has a variety of reserach projects in specific areas of autonomic computing, covering a wide spectrum of their own departments, and many topics of autonomic computing.

The Architecture Based Languages and Environments (ABLE) project is a DASADA-sponsored project, and has several projects involving the use of architecture and specifications for self-adaption. Their 'Acme' architecture description language has found great popularity, within ABLE, within other DASADA projects, and in the community at large.

IBM quotes Carnegie Mellon for a completely different set of projects, namely those on self-securing storage and devices, both under the wing of their Parallel Data Lab. Self-securing storage creates a storage infrastructure that is resistant to attack through the logging of writes and separation of storage from the client OS that reads it. Self-securing devices looks at making devices indepenedently have the ability to secure themselves, and have the ability to note activity around them for suspicious activity.

1.3 Research Groups and Conferences

Large-scale coordinated research efforts in autonomic computing are currenly relatively uncommon, with the exception of the US Government project "DASADA" described below. The related and slightly more mature fields of grid and pervasive computing have formed collaboration efforts, and more formalized research collaborations will likely start to form in autonomic computing as it matures as well. Several workshops have been run on different aspects of autonomic computing, though there is not yet a well-recognized conference focusing solely on it. This will likely change as well, again similar to grid computing and pervasive computing, as the field matures.

1.3.1 Conference Workshops

Autonomic computing has thus far been limited to workshop topics, rather than entire conferences, though several of these workshops appered in 2002 and 2003.

http://www.cse.psu.edu/ yyzhang/shaman/ The Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN), has covered much of the published academic work on autonomic computing. Sessions for the 2002 workshop included reliability and recovery, systems support for self-healing, frameworks for dynamic adaption, self-monitoring, and adaptive networking. The 2002 workshop was hosted in conjuction with the 16^{th} Annual ACM International Conference on Supercomputing.

http://www.caip.rutgers.edu/ams2003/AMS2003.htm The Autonomic Computing Workshop is actually part of the workshop on Active Middleware Services (AMS 2003). It hosts a variety of papers, including autonomic applications, storage, middleware, and architectures.

http://tesla.hpl.hp.com/self-manage03/ The first Workshop on Algorithms and Architectures for Self-Managing Systems, while not often mentioned on its own, features sponsorship from big names: the International Symposium on Computer Architecture (ISCA) and the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS) of the ACM. Topics seem fairly focused on individual uses of autonomic computing rather then general architecture, and there is an entire session dedicated to self-healing.

1.3.2 DASADA

http://www.schafercorp-ballston.com/dasada/index2.html

The Dynamic Assembly for Systems Adaptability, Dependability and Assurance (DASADA) project is a research effort funded by the Defence Advanced Research Projects Agency (DARPA), which is a branch of the United States Department of Defence. Its goal is to create technologies that would allow for the creation and recomposition of systems at runtime, and monitor the system to ensure it functions correctly. This must be done in heterogeneous environments, and must be able to operate autonomically.

The DASADA project consists of several university and industry research efforts into a variety of areas that are important to autonomic computing. A strong level of co-operation

has been shown between the projects, with much of the work becoming interdependent over its term. A structure for measuring and monitoring in terms of 'probes' and 'gauges' has been developed, and a fair deal of sharing of the related technologies for implementing these has taken place.

1.3.3 The Globus Project

http://www.globus.org/

The Globus Project concentrates on creating standardized environments for grid computing. Grid computing has much in common with autonomic computing, in that a grid is distributed, often across organizational boundaries, and distributes workloads across the resources in the system. While this relates most closely to self-optimization, a working grid computing environment by nature is self-healing, and to a lesser extent self-configuring and self-protecting.

The Open Grid Services Architecture is being developed as a standard for grid services, based on many of the recently standardized web services technologies. The Open Grid Services Infrastructure has reached version 1.0, and defines mechanisms for "creating, managing, and exchanging information among entities called Grid Services." [10]. The advantage of this in an autonomic computing environment is that standards are actively being produced, so unlike many of the current corporate offerings, allows for research and reasoning to be applied to the design.

The Globus Project released their 3.0 version of the Globus Toolkit in July 2003. The Globus Toolkit is an open-source set of services for distributed and grid applications, and in version 3.0, is a direct implementation of the OGSI standard. The many communication services that this toolkit provides has led many vendors to base their offerings on top of this.

IBM, being one of the many companies offering their support for OGSA, has developed a middleware layer called OptimalGrid[11], which provides autonomic capabilities in the deployment and execution of a grid application as a layer on top of OGSI. This project uses autonomic computing techniques to hide complexity of a grid computing environment from the end user of such an environment.

HP has also declared their support for the OGSA, contributing significant research effort to the Global Grid Forum (GGF). The GGF is a group of technology companies and researchers dedicated to developing grid technology, and have contributed to efforts on the OGSA and other projects.

1.4 Security in an Autonomic Computing Environment

Security is a key concern to any autonomic system. One of the four key attributes so loved by the media is **self-protection**. Number five in IBM's eight attributes of an autonomic system even explicitly states this.

> A virtual world is no less dangerous than the physical one, so an autonomic computing system must be an expert in self-protection. [1].

This element has been systematically ignored and downplayed by most designers of autonomic computing technologies, as well as those developing them. It suffices for most vendors to state that their systems will, in fact, protect themselves, and rarely provide details on how, or evidence that they in fact can self-protect.

The most notable general document on autonomic computing security is from IBM[5], which describes the general behaviour of an autonomic security system. This is done only from the basic view of an autonomic element and does little to explain the overall architecture. Except for some basic offerings for their storage manager, again focusing on information access control, little has been done to add self-protection to their products.

Academically, there has been a lot more attention paid to autonomic self-protection. Research has focused mostly on systems that resemble modern intrusion detection systems (IDS), and put very little focus on how they could tie into a larger autonomic security structure.

One popular approach in autonomic system security, which fits in well with autonomic computing's biological analogy is that of computer immune systems. Research at the University of New Mexico [12] has applied the concepts found in the human immune system to create a general self-aware artificialimmune system (ARTIS)[13]. The system has been applied in a number of security contexts, including network intrusion detection.

Often paired with the issue of security in autonomic, grid and pervasive computing systems is the concept of trust management. Trust management deals with the specification and enforcement of access, usage and privacy policies, all of which can be considered part of a security policy. In this report, we will focus more on cases where security policy has been already circumvented, and as such will not touch upon trust management except where it relates to the failing of a security policy.

2 Autonomic Security Attributes

In many ways, computer security is faced with much the same problems in an autonomic environment environment as in traditional computing environments. Administrators of a system wish for the system to be able to be used in a certain manner, and attempt to configure the system to prevent it from being used in ways that were not intended. A malicious user wishes to perform an operation not allowed by the administrators, and will do their best to achieve this by any means possible. Often, this can be accomplished by finding flaws in computer software or computer operators.

Some similarities to traditional computing security include:

- Ultimately, there is some collection of humans making security policy decisions, and operationalizing these into computer configurations.
- The threats are similar, and at least for the immediate future, the modes of attack are likely to be the same.
- Some number of flaws in software will always be discovered and used by malicious users before they are discovered and fixed by the software designers. The flaw may be further exploited between the time that a solution is found and the solution is applied to all systems.
- There are inevitable tradeoffs between security and ease of use and performance.

2.1 Security Aspects of Autonomic Systems

Autonomic computing introduces some new features to a system which could affect the potential for security. Some of these features lead to restrictions on what solutions can be applied, while others guarantee capabilities that can be used to the advantage one the implementor of security systems.

Autonomic systems by definition have the ability to reconfigure themselves to certain situations. This means that security implementations have to be dynamic; they must adapt to provide the best possible security for a particular configuration while not interfering with the deployed services, even if those deployed services change. However, the ability to reconfigure provides a tool for autonomic security systems, which can rely on this to configure individual components to reflect the desired level of security.

The decision making elements of an autonomic system could themselves could become a target of attackers. These powerful components, if compromised, could easily be used to instantly apply the attacker's own design to a system. If the attacker gains control of a higher-level autonomic management component, a policy could be put in place to not only compromise all components under its control, but to actively resist any attempts by the original managers to reassert control. A great deal of protection will have to be offered to the autonomic managers themselves to prevent their compromise.

Number seven of IBM's eight attributes of an autonomic computing system states:

> An autonomic computing system cannot exist in a hermetic environment. [1]

Autonomic systems are by definition heterogeneous, and implied by this is the fact that they are distributed. It cannot be guaranteed that all elements of an autonomic system are under the control of those defining security policy. This creates a challenge in that isolation cannot be relied upon for security. Further implied is the fact that a wide variety of systems must be protected, including legacy systems not designed to be autonomic at all. However, this presents an advantage found in many biological systems - that of *biodiversity*. Since an autonomic system has to be able to handle heterogeneous environments, this gives it the ability to prevent or respond to threats by diversifying itself. Some research has experimented on introducing small variances in the same application at compile and load time to provide this biodiversity [14]. Even if an attack successfully takes control of several elements of our autonomic system, it is very unlikely that any one threat would successfully compromise the entire system.

Perhaps the most important aspect of autonomic computing is the 'autonomic' part. Most security systems attempt to hide or close security holes such that they cannot be exploited in the first place. More advanced is the research area of "intrusion detection systems", which focus on detecting when a security violation has occurred. However, the vast majority of these systems only go as far as notifying the administrator of the system. Missing is the ability for the computer to respond to attacks, or to the threat of attack. Adding these abilities for intrusion response will be a key aspect of autonomic computing security.

2.2 Self-Protection and Self-Healing

The area of security in an autonomic computing environment often overlaps with that of self-healing. This serves as an advantage in that security can often benefit from and make use of self-healing abilities, but often suffers where one is used when the other is appropriate.

Self-healing approaches can often be applied to self-protection of systems. In terms of detection, many indicators that may lead the system to believe it is not healthy may also be a sign that the system has been compromised. In the same manner, an attempt to restore the system to a working state may also revert maliciously damaged software to a clean state. For example, a paper on 'self-cleansing' systems[15] demonstrated a periodic cleansing of a system could be used to limit the damage potential of an attacker, using a technique very similar to software rejeuvenation.

However, self-healing and self-protection ultimately are attempting to guarantee different things: self-healing attempts to keep the system operational and meeting the requirements of its users, while self-protection is attempting to prevent the system from doing things it is not supposed to do. These can be conceptualized as the difference between obligation policies and restriction policies . While a failure to meet an obligation may affect the ability to enforce restrictions or vice versa, the two can also occur independently. In fact, to avoid detection, an intruder may rely on this fact, as an unhealthy system could indicate the intruder's presence.

Self-healing is concerned with the system meeting the requirements of its users; these requirements may be dynamic, and are not black-and-white. In fact, it is usually preferable to meet some obligations rather than none at all. Self-protection is not concerned if the system does not meet its goals, as long as it prevents any prohibited actions from occuring. Like self-healing, it it better to enforce some restrictions than none at all.

One difference between self-healing and self-protection is that the success of self-healing can be improved by healing the system to fulfill requirements again. If a self-healing system can heal quickly, it can minimize visible failures to meet obligations. In self-protection on the other hand, once a prohibited action can be performed, it can never be guaranteed that self-protection can recover from this action.

A collary to this is that we cannot accept the notion of 'acceptable levels of intrusion', like is sometimes possible in self-healing. A paper[16] by Mary Shaw suggests that exact software specifications make implemented systems brittle, and that a system may have a fuzzy range between normal and broken that is degraded but 'good enough' for its purposes. This relies on several aspects of software health, namely the assumptions that the system can be more healthy than required by the users, and that there is in fact a gradual transition between healthy and unhealthy. These two assumptions do not always hold in general, and in terms of self-protection often do not. In a system with redundancy and self-healing, this approach can be used for many forms of failure, but is not useful for self-protection since a failure in security policy can happen at any individual component of a redundant system.

Threats to the success of self-healing and self-protection are also very different. Hardware failures often can be modelled accurately with probability distributions, and the failures caused by software bugs can also sometimes be described as random. While failures of components are often not independent of each other, unrelated components in many cases should not be affected by the health of other components in the system. Security violations are most often triggered by the direct actions of a user attempting to cause them. Here, the only thing that can be modelled as random is when a vulnerability is first targeted. An intruder will often purposely rely on the worst case scenario for a failure, breaking many assumptions regarding randomness. Furthermore, the spread of a compromise between components must also be assumed to be worst case rather than independent.

The difference in goals and threats to these goals between self-protection and self-healing makes it important to distinguish between them, but the connection between them also should not be ignored.

3 Structure of an Autonomic System

IBM's autonomic architecture document[4] outlines the basic structure of an autonomic system, and provides a base upon which the structure of an autonomic security system can be reasoned about. While no direct implementation is publicly available to support this, descriptions of similar architectures by many others, such as NEC, imply that such structures are appropriate and will likely resemble the final forms of such a system.

3.1 Autonomic Elements

The basic building block of an autonomic system is the *autonomic element*, as shown in figure 1. An autonomic element consists of an *autonomic manager*, and a *managed element*, formed into what can loosely term as a *control loop*.

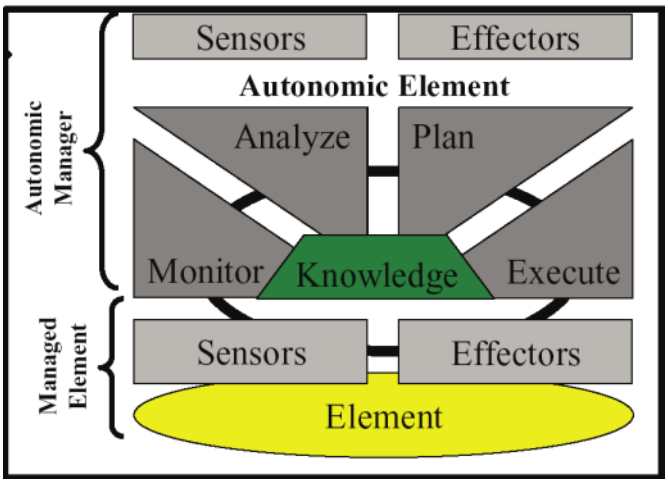

Figure 1: Basic Structure of an Autonomic Element [4]

The managed element could be one of a variety of things. It could be a legacy software component, for example, a web server or FTP server. It could be a component designed specifically for an autonomic element, which provides defined interfaces to ease measurement and control of the managed element. It could even be a collection of other autonomic elements where the sensors and effectors of the main autonomic element would communicate with these lower level components.

The nesting of autonomic elements leads to the idea that a complex autonomic element can be constructed with a nested tree of simpler autonomic elements, as shown in figure 2. In this system, an autonomic manager would manage its managed component by contacting

the autonomic managers of the nested components and coordinating them to perform the required tasks.

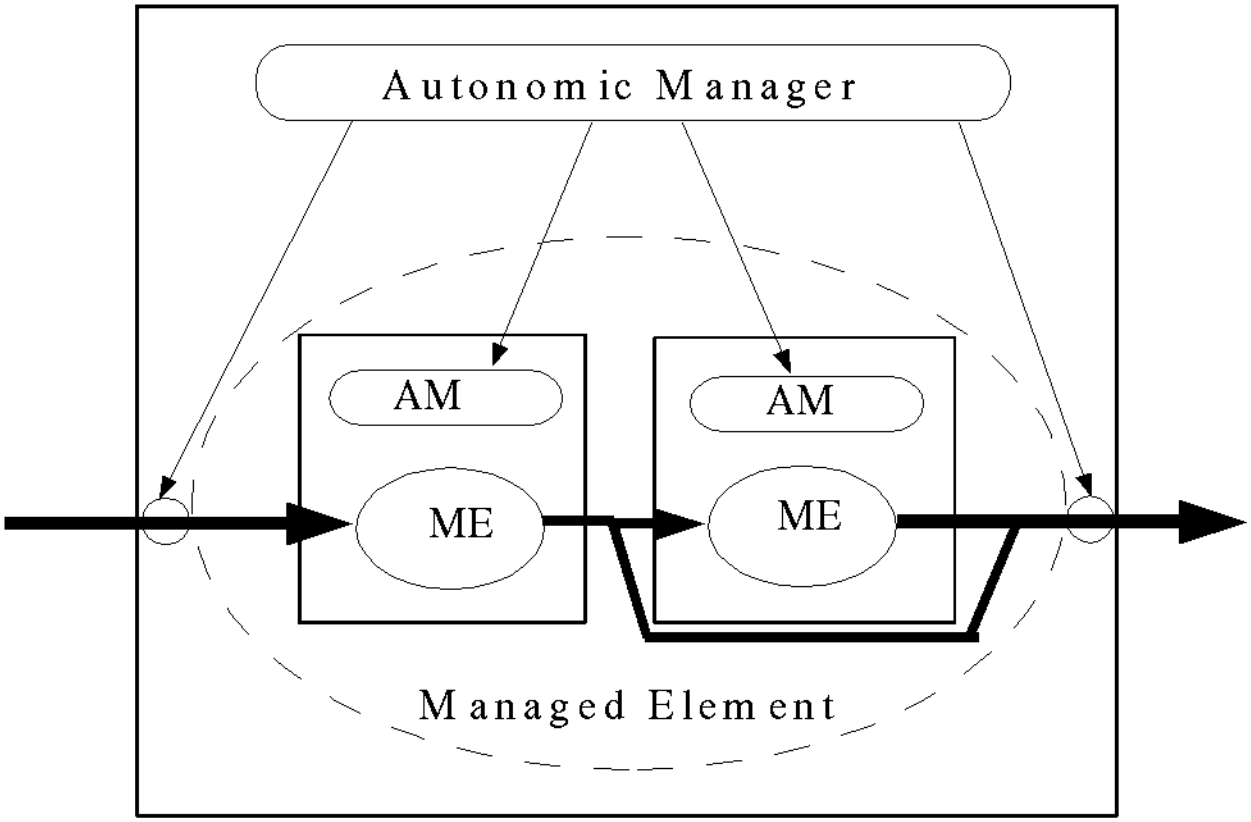

Figure 2: Nested Tree of Autonomic Elements

.

An autonomic manager provides the intelligence to an autonomic system, functioning as a utility-based rational agent. It relies on two connections to the managed element: sensors to observe the state of the system, and effectors to manipulate it. This autonomic manager's implementation may differ dependant on the scale of which it is implemented. On a single computer, this may be nothing more than a process. On a network of computers, perhaps it is a server dedicated to coordinating the network's devices. One could even envision an autonomic manager being virtual, formed by a distributed collection of nested autonomic managers coming to a group consensus about decision - this sort of system would strongly resemble a *co-operative multi-agent system*, and has the advantage of additional fault tolerance. Such a distributed autonomic manager would also have the advantage of being able to be replaced if compromised.

3.2 Layers of an Autonomic System

A system capable of redistributing itself across multiple systems cannot be described in terms of hardware components, or even in terms of software components. Ultimately, these concerns are something to be dealt with by the autonomic system, and ideally something that an administrator would never even need to consider. This leads to the concept that the administrator specifies policy in terms of the autonomic computing system's tasks, and not in terms of any particular hardware or software element.

IBM responds to this by dividing decision making into three different 'decision-making contexts':

- **Resource element contexts** represent individual individual resources or components in a system. This could include a single server, a software application, or any other physical or logical component of the system.
- **Composite resources contexts** represent pools of resource elements. This context allows a resource to be generalized and utilized as a whole rather than as a set of individual devices. It also allows for resources to be shared across multiple systems. An example of this would be a storage context, which could allocate storage based on the requests it has received and the total storage need of the system.
- **System contexts** (described in the IBM literature as 'business contexts') represent the overriding goals and objectives of the system. This is the layer that understands the goals of the system and makes use of the resources available within the system to accomplish them. An example of this would be a payroll system, or a scientific calculation application.

One can view these contexts as a layered system, as shown in figure 3. At the boundary of each layer, policy undergoes a significant translation.

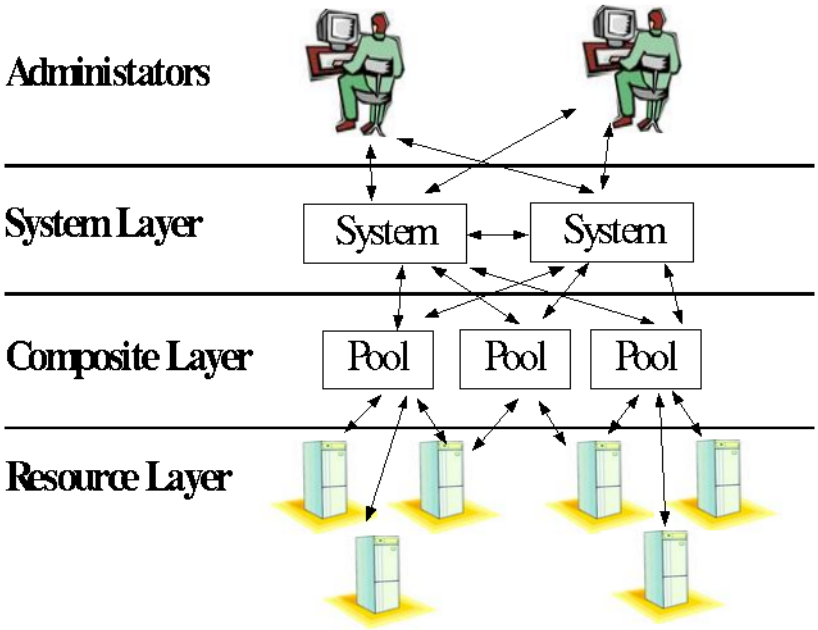

Figure 3: Layered View of an Autonomic System

4 Security Design in an Autonomic System

4.1 System Layer

Security, like everything else in an autonomic system, starts with the administrators with a security policy. This policy is communicated to system contexts through some user interface, and most likely in a very general context. Security policy is not always necessarily specified explicitly as a security policy, but may be inferenced as a requirement of other policies requested by the administrator. A system's basic requirements may require changes to security policy, for example, opening ports on firewalls, or granting access to authentication systems.

Security policies can be associated (directly or otherwise) with each application, plus a 'security system', which would enforce general properties of security that would be independent of any particular system. The autonomic managers of these systems are responsible for dividing up the information and sending it to the correct pools in the composite layer. Furthermore, the system layer must translate policies into forms that will be understood by the composite layer.

Communication may not only originate with the administrators, but with lower levels as well. Important alerts may be sent from the composite layer, which may have itself forwarded these messages as appropriate from individual autonomic elements. The system layer may decide to filter these messages, or based on a high-level policy, instruct the system to take action. Some of these messages, if sufficiently critical, may be sent to administrators through a variety of communication mediums.

Systems will rarely take direct actions to correct security flaws, instead relying mostly on informing administrators of problems or shutting down the entire system in response to a failure. The compromise of a composite pool of resources indicates that every resource element has become compromised, and that all actions taken by the autonomic manager at that layer to correct this have failed. There is little else that can be done at this point unless the system can somehow continue without any resources of that type, however, it is predicted that this will be a rare situation.

While not very effective in preventing intrusions, systems could be essential in detecting intrusions, serving as the communications link between composite contexts. By aggregating reports from multiple composite layer elements, a system can determine better than any individual element the overall state of security in the its system.

4.2 Composite Layer

In the composite layer, policies are received by the system layer. The system only sends relevant policies to a particular composite context. This is where a large amount of conflict resolution will have to occur - systems may have different policies and requirements that affect security decisions, and a composite context will have to take them into account in searching for a solution. In some cases, a set of requests may not even be possible to fulfil, in which case the composite layer must report such a case to the relevant systems.

In cases where policies from different systems conflict, it may be possible that both can be satisfied by fulfilling the request with separate resource elements. A good example of this is the requirement of a system for an amount of secure storage space. The composite context may have normally chosen to distribute the allocation among several shared storage devices, as well as several general-purpose servers under the control of the composite layer, but with the security requirements in place, it may instead choose to take exclusive control of one server for this storage, and disallow any further use of the server by other composite contexts.

Most security policies required by a system context will be relayed to every resource element used to operate the system. The individual resource elements will be required to take direct action to satisfy these policies.

The composite layer has a great deal of ability to detect security breaches. Unlike resource elements, composite contexts are able to aggregate data from multiple nodes. More knowledge can be inferred if suspicious behaviour is seen from multiple resource elements, or perhaps even more suspicious when only seen from one resource element. Reports of possible intrusions from all resource elements of a specific type could often indicate vulnerabilities in that type of element being exploited.

As well as detection, the composite layer has the ability to respond to intrusions. Possible responses in this layer could be to remove a compromised resource element from the pool of resource elements, and redistribute its former workload to other resource elements, to distribute a security patch to fix a known problem, or to notify resource elements to tighten security policies temporarily to outlast an attack.

4.3 Resource layer

While a few decisions have been made at higher layers, it is ultimately up to the resource layer to protect its individual elements.

Ultimately, a resource element is nothing more than an autonomic element, as shown in section 3.1. In most cases, this autonomic element will be a piece of hardware, such as a server, or router. In many cases, the autonomic element may have many embedded autonomic elements - in a server, for example, these could represent individual software components.

IBM envisions an autonomic element as being able to affect the managed element from a security standpoint by control directly into the managed element, as well as control over the flow of data into and out of the component, as shown in figure 4.

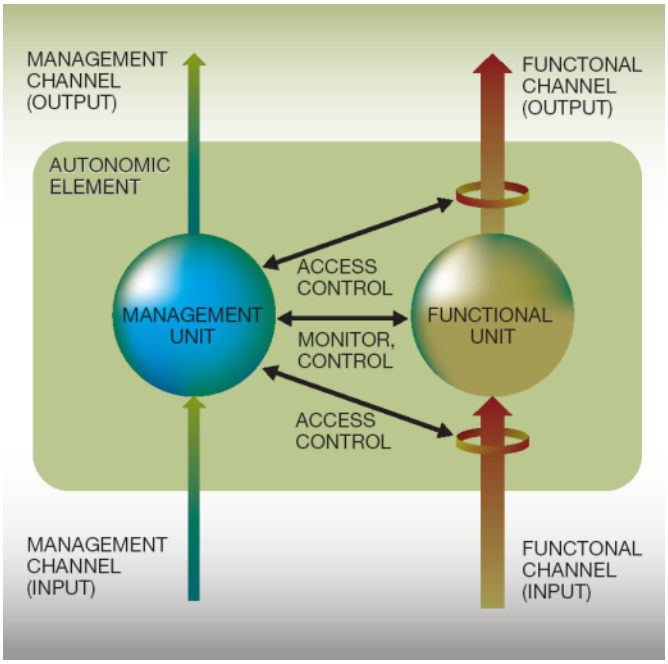

Figure 4: Security View of an Autonomic Element [5]

This diagram was designed with information security in mind: the autonomic manager being given the capacity to block information that should not be allowed to enter or exit the managed element. However, the approach is in no ways limited to information security. When applied to control components, a wide variety of policies can be efficiently enforced.

For example, one can treat the operating system kernel as a managed element, and have an autonomic manager that monitors system calls entering the kernel, filtering out those which would if successful defy the security policy. The BlueBoX[17] intrusion detection system is an example of one such system that does exactly this for the Linux kernel, though there are many other examples.

Here, a hierarchical set of embedded autonomic elements becomes a more important concept. Within a system, a process acts as the autonomic manager for the machine, tracking what

resources are available within the machine, and what resources can be utilized for particular tasks. It is also be a single contact point for the composite layer. This autonomic manager would be capable of starting processes on the machine, such as a web server, or a file replication service. Each of these processes would be started, possibly with its own autonomic manager. If these processes have an autonomic manager, that autonomic manager will communicate directly (and solely) to the autonomic manager that spawned it.

Autonomic managers at the resource level have a large number of security duties, as outlined below.

4.3.1 Preventative Measures

Autonomic managers should take measures to pro-actively enforce security. These will in most cases not be done in response to any intrusion, and at times not even with an explicit mandate from another layer (though of course, it must be within the allowed bounds of the policy).

The key to preventative measures is not interfering with other aspects of the system. For example, an automated virus scan should not use the majority of the processing power on a server dedicated to a parallel computation. Another example is a firewall - the autonomic manager must be careful to open ports for services it is running publicly on the machine.

Some examples of preventative measures include:

- Applying security patches to software periodically, with the assumption that there is a secure source to retrieve these from. Other autonomic elements can temporarily replace the resource if it cannot patch while running.
- Installing, monitoring, and updating firewalls to suit whatever services the autonomic element provides.
- Periodically scanning for viruses (if the security policy allows access of the data to be scanned), and verifying that key executables have not been changed.
- Checking security aspects of programs before they are installed. Techniques that can detect potential security flaws in software executables can warn higher-level autonomic managers of these flaws, and potentially could result in a more secure version being deployed instead.

- Diversifying executables to prevent attacks that rely on specific binary versions of an executable.
- Training intrusion detection systems that rely on detecting anomalous behaviour. This can only be done if the system can be kept secure while training is in process.

Many of these preventative measures will have to be evaluated and applied at the time a particular resource or service is first deployed to an autonomic element. In certain cases, the decisions about how to service a request may be affected by the security decisions of an autonomic element - in fact, a preventative measure taken to ensure security may well be to refuse a request outright, forcing the higher layer to consider a new solution.

4.3.2 Intrusion Detection

A resource element itself must be capable of determining whether or not it has been compromised. To this end, an autonomic manager must have available to it a number of intrusion detection systems (IDS). These intrusion detection systems perform monitoring of traffic into and out of the element to determine a compromise, and if combined with intrusion response systems, may even choose to prevent the flow of data if it is detected to be corrupt.

No single IDS can hope to fulfil the requirements of every autonomic computing environment for every task. The autonomic manager must therefore have a selection of IDSs at its disposal. This implies the need for modular IDSs that can communicate to an autonomic manager in a standard way its detection capabilities and what resources it requires to operate.

An IDS must have a standard way of communicating knowledge of the security state of the system to the autonomic manager. Many techniques from the artificial intelligence world for representing uncertain knowledge could be used here. Certainty factors and fuzzy logic have been used extensively in agent-based expert systems in the past, and would likely both be very effective here. Each IDS module would minimally have to answer, "has my component been compromised?" Assuming the use of certainty factors, an intrusion detection system could respond -1 to indicate that it is sure that the component is operating securely, 1 to report that the component has been compromised, or 0 to indicate it is uncertain. Any intermediate value would represent some level of certainty between these. An autonomic manager would based on its security policy decide at a certain level of certainty to take action.

More intelligent IDSs could return more information to the autonomic manager, especially

when it has some certainty greater than zero of an intrusion. Several sample predicates include:

- Has an intruder gained access to administrative privileges?
- Is malicious code running on the system?
- Could the intrusion allow an intruder affect other autonomic elements?
- Has an attacker attempted (and failed) to penetrate the system?
- Is a denial-of-service attack in process?

An autonomic manager which prioritizes security could easily run several IDSs concurrently, and combine the data from all of them to gain more certainty of the state of the system. In cases where the state remains ambiguous, an autonomic manager can also temporarily deploy additional IDSs to make a decision.

In nested autonomic elements, data from intrusion detection systems will often need to be communicated to parent elements, or even up to the composite layer, as the manager of each aggregation of elements will be better able to determine the pattern or existence of an intrusion. This flow of information can be both push and pull based, with relevant changes in state being pushed up to higher levels, and requests pulled up when a higher-level element requires up to date information to make a decision.

4.3.3 Intrusion Response

Intrusion response is relevant at all layers, but more so at the resource element layer, where autonomic managers can have the greatest flexibility in response to intrusions.

Assuming that at some level an intrusion detection system has decided with sufficient certainty that an intrusion has taken place, the system must decide how to respond. This will involve a great deal of communication to all layers. Even if a local autonomic element is capable of handling the intrusion, a higher level action may be appropriate in response, for example, if the same intrusion is affecting a large number of elements. This distributed decision making is very much like a co-operative multi-agent utility-based system.

At higher levels, very few options are available to autonomic managers. Most actions at these levels will involve adding, removing, or shutting down resources. If the top-level managers of the resource element layer are not compromised, higher levels can instruct these to take

specific actions to correct the situation. For example, the system layer could determine that a security breach can be solved with a patch, which would get distributed to the composite layer and eventually down to the resource layer. In this case, however, it is ultimately the responsibility of the resource layer to apply these patches.

Autonomic elements at the resource level will have a great deal of flexibility in responding to intrusions, as they have knowledge of the inner workings of the system. Often, a corrupted process can be restored to a checkpoint state or restarted, using many of the techniques that will also apply to the aspect of self-healing. A process could be configured to run in a more secure mode in response to certain ongoing attacks. In some cases, the managed element itself may have been instrumented to allow it to respond to attacks. Autonomic elements controlled by an autonomic manager can be replaced with fresh copies of the element, or even with different elements capable of providing the service.

5 Challenges in an Autonomic Security System

Each of the following sections outlines a critical component in an autonomic security system. The currently available technologies and approaches are listed, as well as potential areas for future study.

5.1 Intrusion Detection

An autonomic system needs to be able to detect intrusions. This is a problem that has been studied for many years outside of the context of autonomic systems, called intrusion detection systems (IDS). These systems vary wildly is design, but in general are classified by the following attributes. For further information on classifying and evaluating IDS, see [19].

- **Host-based vs. Network Based:** Host-based systems reside on the system that is being protected, while network-based systems normally reside on a separate machine dedicated to monitoring the traffic entering and exiting the system.
- **Knowledge-based vs. Behaviour Based:** Knowledge-based systems are designed to detect known bad activity, while behaviour-based systems are designed to learn the patterns of good activity and warn on behaviour it does not recognize. The former is often vulnerable to unrecognized attacks, while the latter tends to generate large numbers of false warnings when legitimate but unrecognized behaviour is initiated.
- **Continuous vs. periodic:** A continuous system runs in real time, analyzing and making decisions on activities as they happen, while a periodic system collects data, possibly from log data, and makes a determination after the fact whether an attack took place.

No one system is sufficient for an autonomic security system, as each system has a complex set of tradeoffs. Therefore, an autonomic system should have a variety of IDSs at its disposal; the autonomic system should be able to deploy zero or more of these systems as appropriate to the security policy and situation. This further implies that there needs to be a standard way of installing and uninstalling (or at least activating and deactivating) an IDS, and a standardized way of communication with a management unit.

The Intrusion Detection Message Exchange Format (IDMEF)[20] is a standard being prepared by the Intrusion Detection Exchange Format Working Group of the Internet Engineering Task Force (IETF). It defines a rich XML-based language for communicating alerts from

intrusion detection systems. Central to the goals of the language are the abilities to extend alerts to contain new information, yet allowing systems that cannot interpret this extended information to still work with a subset of the information. This technology is one example of work that could be used to enable communication between IDSs and an autonomic element, though it still leaves unanswered the question of control in the opposite direction.

5.2 Intrusion Response

An element missing from most IDSs that is required by an autonomic system is that of intrusion response. IBM states that when dealing with intrusions, "we must assume that it occurs before we are aware of it" [5]. This point is debatable - several intrusion detection systems, in the most trivial case a firewall, can block an attack before it succeeds. However, we can't hope to block all attacks with these systems, so the system must also have a variety of responses available for use if it is detected that there has been a security violation after the fact.

This is, in the most general case, the set of effectors an autonomic manager can use in response to a detected security problem. A wide variety of responses must be available to any particular autonomic manager to give it the flexibility to choose the best response.

One very appealing aspect of intrusion detection that could be used for intrusion response is the concept of *biodiversity*. Researchers at the University of New Mexico have proposed systems[14] capable of adding random variations at compile time to make executable binaries less vulnerable to attacks depending on the exact layout of code in memory. While useful, an autonomic computing environment has much more flexibility to make sweeping changes. One could imagine having multiple pieces of software capable of fulfilling a particular system requirement; while one may be favoured over others for performance or feature reasons, in the case where this preferred software becomes a target for intruders, a different software implementation could be used. At a sufficiently large scale, work could be moved to machines with entirely different operating systems if the need arose.

Other work in intrusion response has often combined intrusion response with intrusion detection, allowing for attacks to be blocked as they occur. An example of this is work on system call delays [18], where anomalous system calls from a process to the operating system are delayed before being allowed to complete. It was found that the pattern of system calls become anomalous so quickly during intrusions that the system calls are delayed sufficiently to prevent the attacker from getting access in any reasonable amount of time.

Work in autonomic computing relating to self-healing can also be used for self-protection. For example, if a program can be retrieved from a secure source, a corrupted or infected software element could be restarted quickly using recursive restart techniques [9] described by the Recovery Oriented Computing project. In general, the ability to stop and start services is a powerful tool that can be used to thwart intruders.

5.3 Security Policy Representation

The security policies at every level of an autonomic system need to be represented in a common format that can be stored and communicated between autonomic managers.

There is much work going on for domain-specific security policy representations. An example is the work on the OASIS Extensible Access Control Markup Language (XAMCL) [21] to represent access control policies in a standard XML-based format. Another example is the World Wide Web Consortium's Platform for Privacy Preferences (P3P) [22], which is a standard for communicating privacy preferences and policies between a provider of private information and the consumer of such information. While these tools are useful, a generalized way of representing security concerns, capabilities, and policies needs to be designed in such a way that the information can be reasoned with to make security decisions.

One such language that has had significant success in recent years is the Ponder[23] policy specification language. While its initial roots seem to imply a data security centric model, Ponder has been successfully applied both inside and outside the Policy Research Group at Imperial College where it was invented, for many forms of policy based management, from security to data location. The language has shown itself useful as a tool for general information technology systems management, and as such, could be useful for autonomic security, or even as a policy language for autonomic computing systems as a whole.

5.4 Operationalization of Security Policies

An administrator of an autonomic system does not communicate security policies in the form of firewall configurations and access controls, but in the terms of business policy. The administrator states who can access what resource with which application, and it is up to the autonomic system to implement this. Furthermore, multiple applications may share the same physical resources, and the integration of multiple disparate security policies is also necessary. This is referred to in organizational terminology as the 'operationalization' of goals.

Methods need to be developed for security policies to be operationalized between the system and composite layers, and then the composite and resource element layers. This requires autonomic managers that understand the implications of a security policy in terms of their system, and has the effectors to implement it.

Operationalization of security policies could result from policies that are not initially security-related. For example, a request to redistribute a mail server's functionality to a new server will require a change in the firewall configuration of the affected machines to allow the new traffic. A security policy in this case could even limit the possible set of configurations; for example, an application that has a security policy that requires its data to be kept on a private storage device cannot have its storage device shared in an attempt to optimize resources.

5.5 Autonomic Manager Communication

With multiple autonomic managers communicating in a distributed fashion, a standardized way needs to be determined for these elements to communicate with each other.

The challenge in this is more the determination of what autonomic managers would actually want to say to each other. Once that is in place, a standards body can define a standard for the exact format of these messages. Most vendors have declared their support for web services standards as this method of communication, though specific services also have their own communication forms (for example, XACML also includes a method to communicate authorization requests and responses).

In one example, IBM's OptimalGrid software makes use of TSpaces, an IBM variant of the 'tuplespace' concept for communication between entities in a distributed system. IBM has also stated that OptimalGrid will support the Open Grid Services Infrastructure as a form of communication. Many other vendors have stated that their systems may at the lowest levels communicate using the OGSI specification.

5.6 Security Decision Making

Given a set of data from intrusion detection systems, other autonomic managers, stored security policy, and a memory of previous events, an autonomic manager must decide on a particular action to take. This task is essentially the construction of what is called a "rational agent" in the artificial intelligence field.

Rational agents in these contexts are in a dynamic environment, meaning that their environment changes in time while it runs. There is also the issue of other autonomic managers and their components, some of which may be cooperating with you while others may be compromised and working against you. The environment is often unpredictable and difficult to model on a large scale due to it's probabilistic nature. This is a difficult problem from an AI perspective.

The full set of AI tools may be useful here, from neural networks trained to a specific environment, to an expert system programmed with the best responses for a particular device, to genetic algorithms attempting to find the best solution. having derived a model of the problem. Modelling in particular can often be an important tool to determine how closely a result could approach an ideal goal. These systems closely resemble a "utility-based agent".

The Cooperative Intelligent Real-Time Control Architecture for Dynamic Information Assurance (CIRCADIA)[24] system demonstrates the ability for a system to make real-time tradeoffs between objectives in a system. The system can given a model of the system, possible modes of failure, and a list of weighted objectives, come up with a plan to react to possible occurrences. The sample problem for this system is described as the desire to prevent an attacker from successfully creating an account on the system while at the same time keeping logging output to a minimum (where logging output is required to respond to the attack). Depending on the relevant weights of the objectives, the system can derive plans to handle possible situations.

5.7 Self-Knowledge

Autonomic computing, especially self-protection, requires the system to know about its own operation. This is critical in the formation and operationalization of security policies, as well as determining what intrusion detection and response methods are available (this further implies that the system must know it's own abilities for intrusion detection and intrusion response as well).

Several of the DASADA projects are performing ongoing research to address this issue in the context of autonomic systems. The Achitecture Based Languages and Environments[25] (ABLE) project of Carnegie Mellon University has attempted to implement software capable of self-repair and self optimization using information derived from architecture specifications [26]. The Kestrel Institute has followed another path, in using software specifications to determine how one can interpret sensor information coming from a running system.

5.8 Software Instrumentation for Security Management

While support for legacy managed components is important, there is no reason that future systems should not attempt to actively aid the autonomic manager in detecting and resolving security issues. This will involve modifying components to make them self-aware and actively communicate to the autonomic manager its state and potential problems. This has the added advantage that the managed component may be better aware of states it should be in, and better equipped to make changes to its state without interfering with its operation.

The simple network management protocol (SNMP) is an example of an old technology for exactly this purpose, allowing a management tool to view and modify selected attributed in a hardware device (or software component, though this use is much less common). More recently, Java Management Extensions (JMX) have added similar functionality to Java applications, extending on the concept of JavaBeans. Both of these, however, require explicit support to be added to the component in question.

Tools for "code instrumentation" can be used to automatically add support for security in existing applications where source code is available. Language developments, such as AspectJ, could even allow security aspects to be developed separately in a modular and reusable fashion.

6 Proposal for Future Work

Many areas need to be explored in order to successfully create an autonomic security environment. However, without a basic autonomic computing structure, any attempt to experiment with autonomic computing security constructs would be at best incomplete. Therefore the first and most pressing task is to create an environment in which autonomic security structures can be experimented with.

Once such an environment is in place, the structures described in this report can be tested in some basic scenarios to determine their effectiveness. It will be required to demonstrate the working operation of the structures before anything more complex can be built on top of them.

Finally, experiments on the effectiveness of particular security techniques will be able to be performed. One of the least explored areas as perceived by the author is that of intrusion response. This area will need strong attention in autonomic systems, as in contrast to traditional infrastructures, an administrator cannot be relied upon to handle all intrusions. The concept of security through diversity is particularly appealing, as it can take advantage of the underlying abilities of autonomic computing to make the system more secure.

6.1 Preparing an Autonomic Computing Environment

It has been shown that many of the components of an autonomic security system are individually available. One of the biggest challenges that remains is the act of integrating these components. One of the most pressing issues is the construction of the autonomic managers themselves which will be responsible for making security decisions and taking actions, and form the core of the autonomic computing infrastructure.

The lack of any complete autonomic computing environments makes it difficult to integrate components and experimentally measure their effectiveness in an autonomic computing security structure. Some vendors have promised full autonomic systems as early as October 2003, while others have stated that the entirety of their platform will not have been released until early 2006. Still others predict that a standardized inter-operable autonomic computing architecture is still ten years out.

Another issue is that of openness - to perform research on different structures, one needs the ability to modify and extend these structures; either through direct access to code, the ability to develop module or plugin style extensions, or in the case of Java, sufficient

documentation to make use of classes in an autonomic system. Many of the vendors releasing such technologies may not be inclined to provide such a level of openness.

It is proposed that efforts be directed towards creating an infrastructure in which autonomic computing security techniques can be experimented with. Several approaches could be taken to accomplish this, and time should be dedicated towards evaluating each of these approaches before any one approach is committed to.

One could take an existing autonomic computing infrastructure and modify it to add autonomic security features. This is problematic simply because there are so few systems that could possibly be extended for this purpose. IBM's OptimalGrid package, being based in Java, may be a target for extension, but admittedly is nowhere near implementing a full general purpose autonomic computing infrastructure. Columbia University's Kinesthetics Extreme package could be experimented with as well to see if it provides a sufficiently complete base for autonomic computing.

Alternatively, one could create one's own autonomic computing infrastructure. This would be a giant undertaking, but could be aided by making use of several technologies already available in the areas of distributed computing, (multi) agent systems, and grid computing. Many vendors have already declared their support for the Open Grid Services Infrastructure (OGSI)[10] for exactly this reason. The Globus Toolkit is a working open-source Java-based implementation of this standard for grid computing, and it could be used to provide a base on top of which autonomic computing structures could be developed with much less effort.

Finally, one could potentially create a simulator to simulate the actions of an autonomic computing system. This has a disadvantage in that it could entail a significant portion of the work required to create a working autonomic system, yet make it very difficult to integrate components that already exist into the system. It would be difficult to use such a system for security modelling since one would have to simulate both processes with security flaws and intrusion detection systems to detect the errors, as well as model intruder activity. However, the advantage is that autonomic computing systems on a scale much larger than can reasonably be constructed in a lab can be modelled and their behaviour quickly determined.

6.2 Security Research

Having a base autonomic system will allow security in autonomic computing to be investigted. The first goal should be to investigate how (or even if) the patterns described in this report can be applied to the autonomic system chosen. These patterns must be made more

specific to map them to the specific design-level components of this system, and may require revision, or simplifications for experimentation purposes.

Once this structure is in place, it should be simple to experiment with various forms of security systems. Areas where different systems could be experimented with include the security decision making and control, intrusion deteciton systems, and intrusion response systems. If the autonomic system is sufficiently flexible, it may even be possible to experiment with variations in the self-protection architecture itself.

6.2.1 Security through Diversity

A security advantage that an autonomic computing system has in terms of security is that the assumptions that are made about the structure and components of such a system grant us much greater flexibility. It can be assumed that the system has the ability to reconfigure itself for different situations. It can be assumed that software can be deployed and redeployed as required. It can be assumed that the system supports a heterogeneous set of components. All these assumptions grant us flexibility to change the system.

Returning to autonomic computing's biological analogy, one can refer to this concept as 'biodiversity'. A paper[14] examining how diversity in a computing environment can improve security observed how even small amounts of randomness in the compilation of a binary would make it much harder to exploit common security exploits. However, this work was limited to compile and load time changes, with the assumption that the behaviour of the software had to be identical.

An autonomic system is much less limited, in that it could apply these principles at a much larger scale, in terms of entire programs, entire servers, or even entire networks. Biodiversity at much larger scales would serve as an effective generic response to almost any form of security attack. For example, if an Microsoft IIS web server is hosting a PHP application, and these web servers start getting compromised, it is quite possible for an Apache web server to replace the IIS web server temporarily, as each web server could perform equivalent functions (though, notably, many of the features and internal characteristics may be quite different), and it has already been assumed that the system can configure the web server appropriately.

Research should be done to answer several questions:

- Can autonomic elements in general be replaced by ones that perform a similar function but with differing implementations?

- Is diversity an effective response to prevent the spread of a security compromise?
- How well can an autonomic system determine the scope of a compromise and make a decision as to what to replace?
- Would such a system detect and respond to attacks sufficiently fast to make the approach worthwhile?

6.3 Timeline

I will be doing a co-op work term with the University of Waterloo between September and December of 2003. This provides an opportunity to investigate the issues raised in this report, however puts a rather strict time restriction on what can be developed. The following schedule of investigations are suggested.

Existing System Investigation (3 weeks): An in-depth look at the technologies that could potentially act as a base for self-protection research. Several technologies are available, but the selection of any particular technology depends on the ability to freely use and experiment with the technology, as well as the extra development effort required to use it.

- Columbia University's Kinesthetics eXtreme (KX) project, preferably a setup similar to the demo of their 'Workflakes' system demo described in [27].
- The Globus Toolkit 3.0, building an autonomic system on top of grid technologies, similar to the IBM OptimalGrid project.
- IBM's OptimalGrid project.
- Autonomic structures on top of mobile agent coordination systems, such as open-source Cougaar project or the University of Massachusetts Little-JIL project.
- Autonomic structures on top of an active middleware service.

Analysis and Decision (1 week): An approach to implementing a usable research system will be chosen and documented.

Construction of Autonomic System (4 weeks): Once a system is chosen, a test environment must be created with autonomic capabilities and the shell of a self-protection infrastructure.

Test Application and Exploits (2 weeks): At least one application must run in this test environment, and if the autonomic computing system (and time to experiment) permits, multiple applcations should be deployed. Time should be spent to find some potential methods of intrusion to use as test cases on the system.

Experimenting with Self-Protection (4 weeks++): The ultimate goal, namely to experiment with self protection mechanisms themselves. While this work term will only allow a certain amount of time to experiment with these, the availabilitiy of an autonomic computing system will allow easy continuation of my work during and after my final undergraduate term. My primary focus will likely change in the next three months, but my initial target for this area will be the above-described challenge of intrusion response with diversity, hopefully at the level of software components. However, many things could be experimented with, and one area that will require attention during this time frame will be intrusion detection systems in the context of autonomic systems, as at very least one will be required to trigger the autonomic system to respond.

Documentation and Extra Work (2 weeks): Cleanup and slack for time overruns.

References

[1] *Autonomic Computing: IBM's Perspective on the State of Information Technology,* IBM Research, Westchester County, N.Y., 2001.

[2] "Fujitsu Siemens Computers' concept for Autonomic Systems," *Fujitsu Siemens Computers,* http://www.fujitsu-siemens.com/atbusiness/1020071.html (current July 2003).

[3] A.G. Ganek and T.A. Corbi, "The dawning of the autonomic computing era," *IBM Systems Journal,* vol. 42, no. 1, 2003

[4] *An architectural blueprint for autonomic computing,* IBM, Hawthorne, NY, 2003.

[5] D. Chess, C. Palmer and S. White, "Security in an autonomic computing environment," *IBM Systems Journal,* vol. 42, no. 1, 2003.

[6] "Microsoft Announces Dynamic Systems Initiative," *PressPass - Information for Journalists,* http://www.microsoft.com/presspass/press/2003/mar03/03-18DynamicSystemsPR.asp (current July 2003).

[7] A. Wohl, "Sun's N1 And A New Kind Of Systems Management," *Amy D. Wohl's Opinions*, http://www.wohl.com/wa0274.htm (current July 2003).

[8] "The Berkeley/Stanford Recovery-Oriented Computing (ROC) Project," http://roc.cs.berkeley.edu/ (current July 2003).

[9] G. Caneda et al, "Reducing Recovery Time in a Small Recursively Restartable System," *Int'l Conf. on Dependable Systems and Networks (DSN-2002)*, 2002.

[10] *Open Grid Services Infrastructure (OGSI) Version 1.0,* Open Grid Services Infrastructure Working Group, 2003.

[11] J. Kaufman et. al., "OptimalGrid – autonomic computing on the Grid", *IBM developerWorks* article, San Jose, CA, 2003.

[12] "Computer Immune Systems,", *UNM Computer Science,* http://www.cs.unm.edu/~immsec/ (current August 2003).

[13] S. Hofmeyr and S. Forrest. "Architecture for an Artificial Immune System," *Evolutionary Computation 7*, Morgan-Kaufmann, San Francisco, CA, 2000.

[14] S. Forrest, A. Somayaji and D Ackley, "Building Diverse Computer Systems," *Proc. 6th Workshop on Hot Topic in Operating Systems*, Computer Society Press, Los Alamitos, CA, 1997.

[15] Y. Huang and A. Sood, "Self-Cleansing Systems for Intrusion Containment," *Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN),* New York City, NY, 2002.

[16] M. Shaw, ""Self-Healing": Softening Precision to Avoid Brittleness," *Workshop on Self-Healing Systems (WOSS'02),* Charleston, SC, 2002.

[17] S. Chari, and P. Cheng, "BlueBoX: A Policy-driven, Host-Based Intrusion Detection system," *Internet Society's 2002 Network and Distributed System Security Symposium (NDSS'02),* San Diego, CA, 2002.

[18] A. Somayaji and S. Forrest, "Automated Response Using System-Call Delays,", *Proc. 9th USENIX Security Symposium*, Denver, CO, 2000.

[19] D. Alessandri et al, *Towards a Taxonomy of Intrusion Detection Systems and Attacks,* Research Report RZ 3366, IBM Research, Zurich, Switzerland, 2001.

[20] *Internet Draft January 30 2003, Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition,* Intrusion Detection Working Group, Internet Engineering Task Force, 2003.

[21] *eXtensible Access Control Markup Language (XACML) Version 1.0*, OASIS, 2003.

[22] L. Cranor et al, "The Platform for Privacy Preferences 1.0 (P3P1.0) Specification", *World Wide Web Consortium*, http://www.w3.org/TR/2002/REC-P3P-20020416/ (current July 2003).

[23] N. Damianou et al., "The Ponder Policy Specification Language," *Proc. Policy 2001: Workshop on Policies for Distributed Systems and Networks,* Bristol, UK, 2001.

[24] V. Ha and D. Musliner, "Balancing Safety Against Performance: Tradeoffs in Internet Security," *36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, Big Island, HI, 2003

[25] "ABLE Project Home Page", *Architecutre Based Languages and Environments,* http://www-2.cs.cmu.edu/~able/ (current August 2003).

[26] D. Garlan, S. Cheng and B. Schmerl, "Increasing System Dependability through Architecture-based Self-repair,", *Architecting Dependable Systems,* Springer-Verlag, 2003.

[27] G. Valetto and G. Kaiser, "Using Process Technology to Control and Coordinate Software Adaption," *Proc. Int'l Conf. on Software Engineering,* Portland, OR, 2003.