Security in an autonomic computing environment

by D. M. Chess
C. C. Palmer
S. R. White

System and network security are vital parts of any autonomic computing solution. The ability of a system to react consistently and correctly to situations ranging from benign but unusual events to outright attacks is key to the achievement of the goals of self-protection, self-healing, and self-optimization. Because they are often built around the interconnection of elements from different administrative domains, autonomic systems raise additional security challenges, including the establishment of a trustworthy system identity, automatically handling changes in system configuration and interconnections, and greatly increased configuration complexity. On the other hand, the techniques of autonomic computing offer the promise of making systems more secure, by effectively and automatically enforcing high-level security policies. In this paper, we discuss these and other security and privacy challenges posed by autonomic systems and provide some recommendations for how these challenges may be met.

As computing systems have become more complex, more interconnected, and more tightly woven into the fabric of our lives, the resources involved in managing and administering them have grown at a steadily increasing rate. As the costs of system hardware and software have leveled off or decreased, the costs of the human resources devoted to system administration have continued to grow, and therefore constitute a steadily larger fraction of information technology (IT) costs. The autonomic computing initiative is aimed at addressing these increasing costs by producing computing systems that require less human effort to administer; systems that, like the biological systems that keep our hearts beating and our body chemistry balanced, can take care of routine and even exceptional functions without human intervention.[1]

Like any other significant computing system, autonomic systems need to be secure. Building secure autonomic systems is a challenge for a number of reasons. Many autonomic systems will use new techniques and new architectures whose security implications are not yet well understood. Autonomic systems should not rely on anomalous behavior caused by security compromises being noticed by humans, if they are to benefit from reduced human administration costs. Because many autonomic systems are expected to deal with a constantly changing set of other systems as suppliers, customers, and partners, they need flexible new methods for reliably establishing trust, detecting attacks and compromise, and recovering from security incidents. Because some autonomic systems deal with personal information about individuals, they need to be able to represent and demonstrably obey privacy policies required by national laws and business ethics.

Successful autonomic systems will need to be self-configuring, self-optimizing, self-protecting, and self-

healing. Although security concerns are most obvious in protecting the system from attack and in recovering from the effects of attacks, security will be key in all other aspects as well. Systems must be secure in every configuration into which they might put themselves, and in every state into which they might optimize themselves. Systems must be robust against attempts to provide them with false or misleading information that might lead them to configure or optimize themselves insecurely, to enter into an unjustified trust relationship, or to fail to protect adequately against a malicious attack.

Autonomic computing will not reinvent computer science *ex nihilo* and the security of autonomic systems will not be an entirely new kind of security. All the traditional issues familiar to computer security researchers will arise in autonomic systems, some in more complex and urgent forms. And just as widespread program sharing and ubiquitous network connectivity took computer viruses and worms from a theoretical possibility to an annoying oddity and then to a major security concern, we should expect that the new computing environments made possible by autonomic computing will give rise to unique security threats of their own.

At least as significantly, the new abilities offered by autonomic computing will also include ways to make our systems more secure and our private data better protected. Building and administering secure computing systems is well known to be a difficult task; properly configuring a complex system to conform to security policies specified at a high level is extremely challenging even for skilled practitioners, and even the best-administered computing systems generally conform only approximately to their putative security policies.[2] By automating the process of configuring, optimizing, and protecting systems according to explicitly stated security policies, autonomic systems offer us the opportunity to do better.

In the next section of this paper, we provide a very brief overview of some of the architectural features that will be important in the design of autonomic computing systems, with an emphasis on the aspects of that architecture that relate to security. In successive sections, we survey a number of old and new security issues and opportunities as they apply to autonomic systems, and we describe two existing systems in which some of these issues have begun to emerge. Along the way, we will note both the challenges and the opportunities in providing security for autonomic computing systems.

Architectural features of autonomic computing

Autonomic computing will have implications for computing systems at all scales, from single devices to the worldwide networked economy. At small scales, we anticipate that the units of autonomic computing, generally referred to as "autonomic elements," will be comparatively simple and of fixed function, performing the same activities in concert with the same set of other elements for long periods of time. At higher levels, however, we expect that many autonomic elements will function in a very dynamic environment, in which only the element's essential mission and governing policies will remain constant. The details of how they carry out their mission and what other elements they interact with may change every day, or even every second.

We anticipate that one very common architecture for an autonomic element will involve two parts: a *functional unit* that performs whatever basic function the element provides (such as storage, database functions, Web services, and so on), and a *management unit* that oversees the operation of the functional unit, ensures that it has the resources that it needs to perform its function, configures and reconfigures it to adapt to changing conditions, carries out negotiations with other autonomic elements, and so on.[3] Figure 1 shows a simple conceptual diagram of an autonomic element consisting of a functional unit and a management unit.

The thin arrows connecting the management unit to the world outside the autonomic element represent the management unit's dealings with other autonomic elements (and potentially with other external resources). The thick arrows connecting the functional unit to the outside world represent the channels by which the element acquires the resources that it needs to carry out its basic function, and by which it delivers the results of that function to other elements. The arrows between the management unit and the functional unit represent the sensors and effectors by which the management unit monitors and controls the functional unit, and the arrows between the management unit and the loops around the function arrows represent the access control that the management unit exercises over these functional channels. No autonomic element or other entity can provide a resource to this element, or obtain any ser-

vice from it, without the permission of the management unit, as negotiated and obtained through the management channels.

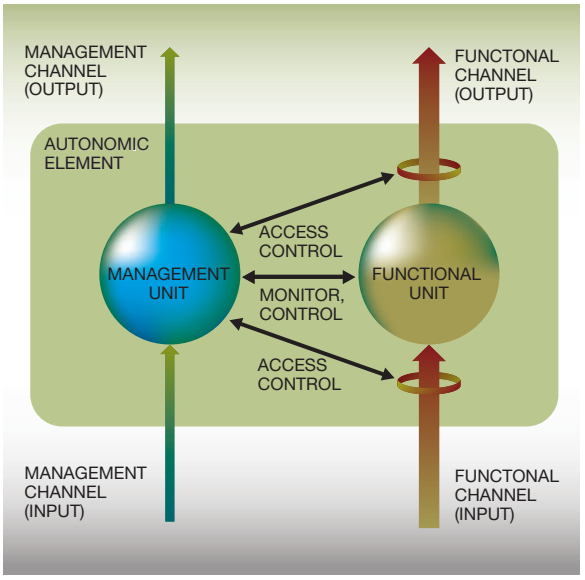
In order to make the decisions necessary to properly oversee the operation of the functional unit and to achieve the flexibility required to make the element self-managing, many management units will carry with them, or otherwise have access to, *policies* that govern and constrain their behaviors at a comparatively high level and *task and state representations* that functionally describe their current mission, strategy, and status at a lower level. Unlike conventional computing systems, which behave as they do simply because they are explicitly programmed that way, the management unit of an autonomic element will often have a wide range of possible strategies available to it in fulfilling the policies that govern it, and an explicit representation of the current state of its efforts to carry out those policies.

Some of the policies that govern an autonomic element will be security policies. An element's security policies may include descriptions of what level of protection needs to be applied to the various information resources that the element contains or controls, rules that determine how much trust the element places in other elements with which it communicates, what cryptographic protocols the element should use in various situations, under what circumstances the element should apply or accept security-related patches or other updates to its own software, and so on. Other policies will control the strategies that an element uses to recover when one of its suppliers fails to provide an expected resource, and to which of its commitments to give the highest priority when not all can be fully met. These policies will either be directly specified by a human, implicitly specified (as by a human accepting a default), or derived from higher-level policies by the rules of the appropriate policy calculus.

Some of the task and state representations that a management unit holds to describe the current status and activities of the element will also be relevant to the element's security. A management unit may, for instance, have:

- A representation of the other elements upon which it currently depends, and how much it trusts each of them
- A representation of the current life-cycle state of the software that the element is running and whether or not there are any security updates available for it
- A list of contact information for one or more other autonomic elements or human administrators who should be notified when certain suspicious circumstances are observed
- Agreements with one or more other autonomic elements to provide it with security-relevant information, such as log-file analyses or secure time-stamping
- A list of previously-vetted resource suppliers, used to quickly verify the digital signatures on the resources they provide

Figure 1 Logical structure of an autonomic element



By explicitly representing both security policies and security-related tasks and states within the system, autonomic elements will be able to automatically handle a wide range of security issues that are currently addressed by human intervention or by comparatively *ad hoc* programmed solutions. Although we do not expect that every autonomic element at every scale will fully conform to this model of management and functional units, policies, and representations, we do anticipate that these design patterns will underlie virtually all autonomic systems above the device level, and they form the conceptual framework for our consideration of the security aspects of autonomic systems.

Traditional and emergent security issues

Every computer security issue familiar to workers in the field will be relevant to autonomic computing. Autonomic elements will need secure authentication protocols to identify each other, secure cryptography to keep their communications from falling into the wrong hands, and secure delegation mechanisms to allow other elements to take actions on their behalf. Digital signatures and nonrepudiation mechanisms will be vital for elements that carry out electronic commerce, and it will be crucial to ensure that the computers on which autonomic systems run are not vulnerable to compromise through buffer overflows, or to service disruption by network flooding. Automatic intrusion detection systems will be more important than ever, since in the absence of direct human control over the operations of the autonomic elements, it will not be possible (or at least not desirable) to rely on human common sense to intuit that something is not right.

The new security issues that emerged in the last years of the twentieth century will continue to apply in autonomic systems. Replicating threats such as computer viruses and worms, and subtler network effects such as routing and feedback loops, will need to be detected and eliminated in autonomic systems at least as urgently as they are in today's systems. A worm that could infect and proliferate between autonomic elements based on a particular implementation library could potentially be at least as severe as a worm that spreads between computers running a particular HyperText Transfer Protocol (HTTP) server.[4] Distributed denial-of-service attacks,[5] in which a large number of systems are compromised remotely and then used in a coordinated attack on a target, would be as devastating in an autonomic system as they are today, if not more so.

The security issues that are urgent today will be even more urgent in a world of autonomic systems. That new world will also bring new security issues of its own, issues that may not be significant or present at all today. At the same time, autonomic technology will offer new opportunities—new ways of securing our systems against attack.

Control, information, and trust

When a system is within a given administrative domain, the owners of that domain have (at least potentially) full control over that system, and complete information about it. In contrast, the owners of a domain have less control over, and less complete and reliable information about, systems in other domains. All other things being equal, a given party will have less trust in systems that are outside its administrative domain, because it can neither completely control, nor completely know, what those systems are doing.

In a dynamic autonomic computing system that spans multiple domains, no single entity has full control over, or full knowledge of, all the components of the system. Each party using the system will therefore have a reduced degree of trust in the system. To ensure that such a system is usable, and used, by the parties for whom it is intended, we must design the system to increase the level of trust that those parties have in the system, so that the net loss of trust is small compared to the gain in efficiency and flexibility from using the autonomic system.

In the current world, businesses and individuals are accustomed to dealing with entities that they neither entirely control nor perfectly trust, as customers, as suppliers, and as partners. In constructing autonomic computing systems that span administrative domains, we must allow businesses and individuals to operate successfully in an environment where critical parts of the IT infrastructure, and of the business and personal infrastructure in general, are similarly under the control of others, and where the details of the relationships with those others can change without human intervention.

Trust is necessary both for the routine operation of autonomic computing systems and for the initial adoption of those systems by customers and users. Not only must the elements of an autonomic system have good reason to trust the other elements that they discover and with which they interoperate, but the human decision makers who opt to use an autonomic system in the first place must have good reason to trust that the system as a whole will serve their purposes. Establishing both of these kinds of trust will require considerable invention. Human decision makers will always prefer to make their own trust judgments in some areas. On the other hand, automated trust establishment in some domains may come to be recognized as more reliable and consistent than *ad hoc* manual trust decisions made by humans.

There are a number of existing mechanisms for establishing and reasoning about trust and trustworthiness, and some of them will have a role to play

in the development of trust in autonomic systems. These range from the decentralized web of trust used in PGP-style (Pretty Good Privacy-style) systems[6] to the strict hierarchy of certificate authorities used in others.[7] An autonomic element might make trust decisions based on a particular trust scheme, depending either on explicit instructions from the humans controlling it, or by a policy derived through the appropriate policy calculus from higher-level human-provided policies.

When one autonomic element relies on information that it gets from another, it should ideally have access to the entire chain of elements that produced that information. On the other hand, if the elements involved are not all in the same administrative domain and do not all have precisely the same goals, the element supplying the information may not be willing (that is, may not be permitted by its own security policies) to reveal the identities of all the other elements in that chain. In such circumstances, how can the element receiving the information determine to what extent the information can be trusted? There will be roles here for technical methods (such as zero-knowledge proofs), for explicit rules (where, say, a human simply reassures the element that certain information sources can be relied upon), for contracts and other legal arrangements (where an element can assume that information can be trusted, because it knows that legal recourse exists if it turns out to be false), and for various kinds of trust calculus.

The security and trust policies that govern an autonomic element will determine how demanding it is when making trust decisions, how readily it trusts other elements, and how much corroboration it seeks before relying on information. These policies will also constrain some of the actions that an element can take. For instance, if one of an element's suppliers becomes unavailable or stops performing acceptably, the element will need to decide which of the potential replacement suppliers can be trusted, both to take over the function of the failed supplier competently, and to be sufficiently trustworthy in that role. By enabling computing systems to make these decisions in consistent and reliable ways, autonomic techniques will engender an extremely adaptive and dynamic operational style, without compromising security.

System compromise vs system outage—coping with intrusions

Autonomic systems will be self-protecting and self-healing. This means, in part, that they will be capable of detecting intrusions on their own, and reacting to them so as to eliminate the intrusion and restore the system to an uncompromised state. To examine how this might be done, we divide an intrusion into three phases:

- Compromise—the initial intrusion and subsequent actions by the intruder
- Detection—the determination, by the system, that an intrusion has occurred
- Restoration—the elimination of the intrusion and restoration of the system to an uncompromised state

In many ways, the compromise of a system by an intruder presents a reliability problem. The system may or may not still function, and it is likely to function in a different and unintended way. But compromise is also different from ordinary unreliability in important ways. In an unreliable system, we generally assume that errors are uncorrelated, in the sense that having two simultaneous errors is much less likely than having one, or none. Compromise can easily violate these assumptions. Attacks are malicious rather than random, and the compromise of several parts of the system can be correlated to further the purposes of the attacker. On the other hand, while some system faults are detectable precisely because of the pattern of failures that they produce, human attackers are motivated to cover their tracks and keep their intrusions inconspicuous. These factors make intrusion potentially more dangerous, and more difficult to detect and cure, than simple reliability problems. Some of the means that an autonomic element uses to deal with an intrusion will be similar to the means it uses to deal with unreliable hardware, or other nonmalicious failures, but other means will be very different.

Systems are usually organized so as to prevent intrusion. Tools such as firewalls, passwords, and access control limit the ability of external parties to act upon a system. At one time, preventative measures were thought to be the solution to the problem.[8] It seems likely, however, that additional protective measures are necessary, both because it is difficult to deploy a fully secure system, and because inevitable design and implementation flaws provide targets of attack even in supposedly secure systems. Fine-grained behavioral restrictions have been used to further limit what a user can do within a system by limiting what actions various programs can take (see for instance the signer-based security model in Java** 2[9]). These can be useful, but they too have

their limitations (for instance, specifying precisely which behaviors should be permitted for a given program can be a complex and error-prone task). Finally, detection systems attempt to notice when an intrusion is taking place, often by noticing actions that should never occur in a normal system, or by noticing when a user's or a program's actions go beyond some statistical measure of normal operation.[10]

As with the detection of virtually any behavioral characteristic of a system, it is impossible to detect with 100 percent certainty that a system has been compromised. Therefore, detecting an intrusion or a compromise in an autonomic system will always be an approximate business. Even the best detection methods will always have false positives or false negatives, or both. Because of the malicious nature of an intrusion, we must assume that it occurs before we are aware of it, leaving the intruder some amount of time to compromise the system. The best an autonomic system can do is to minimize the amount of time between the intrusion and its detection, so as to limit the amount and extent of compromise.

Once we are aware that an intrusion has taken place and have some idea of what parts of the system may have been compromised, we can act. The first thing we must do is to cut off the intruder as much as possible. External communications links to the affected parts of the system can be severed, and affected machines can be taken down. Autonomic elements that have been compromised can be terminated or isolated through changes to high-level policies. If this stops the attack, our remaining problem is restoring the system to an uncompromised state. To do this, we divide the system conceptually into a stateless part and a stateful part. The stateless part has no important state that changes as the system operates, so restoring it is relatively easy. We can restore each system from a known secure backup (perhaps the original distribution set for the system) and bring it up again. Autonomic systems must be able to perform these functions automatically (at least in typical cases), without human intervention.

Restoring the stateful part of the system requires more preparation. The state of the system must be maintained in such a way that (1) corruption can be detected and (2) corruption can be eliminated. One approach to these requirements involves keeping the state of the system in an encrypted, redundant form, distributed across a number of logical and/or physical nodes. Techniques are available for restoring state where as much as one-third to one-half of the copies of the state have been corrupted (see the section "Secure distributed storage," later). In many cases, it will be advantageous to distribute redundant copies of the state in any case and check them against each other periodically, so that random errors can be dealt with. Using encryption and keeping additional copies helps us deal with intrusion in a similarly general way; this is one technique that autonomic elements can use against both malicious and nonmalicious failures.

Once the compromised systems have been restored, they can be brought back on line, their communication channels can be restored, and they can once again operate normally. These operations are currently performed by skilled human operators; automating them so that autonomic systems can perform them automatically in typical cases (requiring human help only rarely) will be a significant challenge.

Fraud and persuasion

The policies that govern an autonomic element's high-level behavior, and the task and state representations that allow it to reason about its own activities, provide high-value targets to a potential attacker. When an attacker compromises a traditional computing system, the attacker may, for instance, insert a piece of code that causes the system to silently send him or her a copy of some important information at a particular e-mail address at a particular time. If that address becomes unavailable, or a network gateway blocks the transmission of the information, the leak will stop. But if an attacker were to compromise an autonomic element, and add to its policy database a policy that required it to provide some important information to the attacker at a certain interval, the autonomic element would then use every resource at its disposal to ensure that the information was delivered. The attacker would have harnessed the element's own ability to adapt to changing conditions and adopt new strategies for the purpose of stealing the desired information. Preventing this sort of high-level subversion will be an important part of the security of autonomic systems.

On the other hand, the security policies that govern an autonomic element can, if properly secured against tampering, provide new levels of resistance to attack. If the attacker in the example above alters only the functional code of the element, or only the task representation held by the management unit, the element will probably not leak its important information to the attacker, because that leakage will

be forbidden by its security policy. Autonomic systems, because they contain explicit computer-readable representations of the security policies under which they operate, can potentially be more resistant to attack and subversion than current systems, which contain only functional code whose behavior may or may not conform to the policies to which humans would like it to conform.

In a preautonomic system (see Figure 2), an attacker who can implant a back door into a computing system can create a data leak that will extract and send any information the back door code can identify. In an autonomic system (see Figure 3), even if an attacker can implant a back door into the programming of the functional unit, the element's management unit will typically block the back door code's attempt to leak data back to the attacker, because the element's security policies will not allow the transmission.

Autonomic elements will make many decisions without direct human instruction, and they will automatically sense and adapt themselves to changes in their operating environment. In these respects, the field of autonomic computing shares features with the only slightly older field of autonomous agents (see for instance Reference 11). Like autonomous agents, autonomic elements will depend for their correct operation on accurate information from other elements and from the outside world. Although there has been some theoretical work on protecting autonomous agents from attacks based on providing them with inaccurate or biased information (see for example Reference 12), this is still a mostly unsolved problem; autonomic elements will have to include safeguards to prevent such deception. In the short term, these safeguards will likely work primarily through policies that instruct the elements to rely only on information derived from sources that a human has explicitly declared trustworthy. As policies become more complex and more flexible, more of the work of making trust decisions can be put into the elements themselves. In fact, this will quickly become a necessity, as the complexity of the system of autonomic elements grows.

Figure 2 Data leak via back door implanted in functional unit

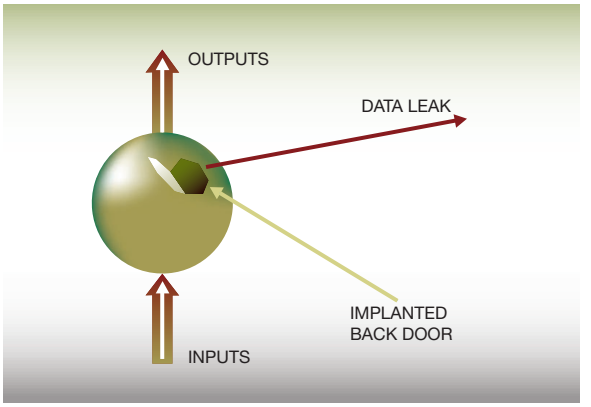
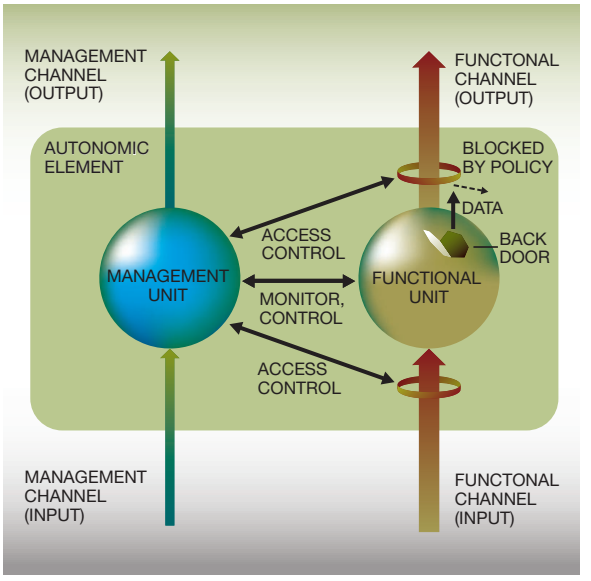


Figure 3 Blockage of back door leak by autonomic system



Privacy issues

One of the factors in the increasing number and complexity of information systems is the increasing amount of information available to be processed. Much of this information is personal information, relevant to some individual who may desire to control how the information is gathered, and how it is used or distributed. Different jurisdictions have different definitions of personal information, give individuals different degrees of legal control over such information, and impose different legal requirements on entities that hold such information. The European Union's Directive on the Protection of Personal Data, the United Kingdom's Data Protection Act of 1988, and U.S. regulations on the privacy of medical records are current examples. Some industry groups

adhere to voluntary guidelines on personal data protection, and individual enterprises often have their own privacy policies that govern the use and distribution of personal information. Contractual relationships may also call for specific handling of personal data.

If autonomic computing is to reach its potential, autonomic elements must be able to gather, transmit, and process personal data in a way that respects the relevant privacy policies, without requiring direct human intervention or assistance. Architecturally, this means that autonomic elements must maintain representations of both the applicable data protection policies and the privacy status of the various kinds of information that they process. It must be possible for an autonomic element to reliably and automatically determine the data protection class of each unit of data that it deals with, and to securely and automatically retrieve the correct policy to apply to that class of data. It must be possible to make routine changes in policies without changing the underlying programming or architecture, since laws and guidelines on personal data protection are in a state of constant flux and are likely to remain so for some time.

Autonomic techniques offer opportunities for increasing system security.

Security and privacy negotiation

It is not enough for an element of an autonomic computing system to ensure its own security or its own privacy measures. Because it depends critically on other elements, it must also be able to trust the security and privacy of those other elements. An autonomic computing system that spans domain boundaries must allow elements to negotiate security and privacy policies, and to gather and securely exchange the information required to verify compliance and to satisfy audit requirements. This will require ontologies and standards expressive enough to allow the specification and negotiation of security and privacy policies, as well as both technologies and standards for record keeping and auditing of behavior relevant to these policies. (Existing technologies in this space, such as Platform for Privacy Preferences (P3P),[13] are currently aimed at allowing individual retail users to specify simple static privacy policies; the requirements for autonomic computing will go far beyond this.) New cryptographic techniques and other technologies will be needed to allow data to be anonymized or aggregated where required, and to be encoded so as to be secure against various new and old types of threat.

Security and privacy policies and their negotiation must be able to take complex political and geographical situations into account. This is one area where geography does not disappear in cyberspace. If an autonomic element has data that, by law, may not enter a certain region of the world, then the elements to which it provides those data must be able to promise, and later perhaps auditably prove, that they did not ship the data to that region, and that each other element to which they provided those data made the same promise and can provide the same audit data. The same considerations apply to nongeographic restrictions. Powerful and flexible ways to specify how data may and may not be processed are needed, as data move through a complex autonomic system. In some applications, it may be necessary for one party to monitor, in more or less real time, the things that have happened to the sensitive data that have been provided to the system. Enabling this in a real system will be a significant challenge.

Further opportunities

Autonomic techniques offer opportunities for increasing system security, both at the level of single elements, and in systems made up of many elements. The key advantage at the level of the single element will be the explicit representation and enforcement of security policies; as we have pointed out previously, system security will no longer rely on high-level security policies being accurately converted into low-level implementation when the system is first coded and configured. Instead, autonomic elements will maintain representations of both the security policies in effect and the tasks to be performed, and will be able to ensure that the actions they take conform to the relevant security policies. With this opportunity come challenges: it will require considerable invention to ensure that the various security policies in use by the autonomic elements in a system are compatible, to verify that the collective behavior that they produce has desirable overall security properties, and to design mechanisms to automatically handle policy conflicts that arise.

Very few autonomic elements will operate in a vacuum. Many groups of elements in an autonomic system will in fact have the same interests, or at least significantly overlapping interests. When a large number of elements have an interest in the security of some part of the system, they may be willing to provide or exchange security-relevant information and to allow it to be aggregated. An element could provide security analysis as a service; any other element willing to provide it with security-relevant data could benefit from analysis of all of the aggregated data, potentially discovering nascent or ongoing attacks that would not be visible from examining only a single data set, or that might not have been distinguishable from innocent traffic. This sort of collaborative distributed intrusion detection is sometimes done on an *ad hoc* basis today; autonomic computing will allow it to become routine. Many other security-relevant services, such as third-party signing, time stamping, and security auditing might be provided as services by elements of an autonomic system.

As presented in the next section, the practicality of automated security response has already been demonstrated in the form of "immune system"-style countermeasures against replicating threats such as viruses and worms. Autonomic computing will provide a sound infrastructural basis for this kind of automated security collaboration.

The policy-management advantages that autonomic computing offers to a single element also apply to groups of elements in the same administrative domain, or otherwise under the control of the same set of policies. Ultimately, the owner of an autonomic system will need to be concerned only with security policies at the level of overall business objectives and operating standards, and the autonomic elements comprising the system will automatically derive and distribute the lower-level policies constraining their detailed operation. In the shorter term, humans will still be involved in the creation of middle-level and even low-level policies, but by securely distributing these policies to all the elements of the autonomic system, significant amounts of effort (and opportunities for error) will be avoided.

Sample products and implications

In this section, we discuss security issues in two existing systems that exhibit various kinds of autonomic behavior.

Immune system. In the 1990s, replicating security threats in the form of computer viruses and worms became a significant problem for the computing community. One effective response to that threat was the development of a biologically inspired "immune system," as described in Reference 14. Although this immune system predates the idea of autonomic computing as we present it here, it was one of the progenitors of that idea, and the architecture and design of the system have autonomic aspects that are worth considering.

Key to the effectiveness of the antivirus immune system is that it automates many of the activities involved in discovering a new virus and protecting computer systems against it. Until the development of the immune system, many parts of the process depended on human action. A new virus (discovered either by heuristics in antivirus programs or by a user noticing odd system behavior) would be manually captured and forwarded to an antivirus company. There it would be analyzed by a human expert using various debugging and analysis tools. That expert would then develop a virus "definition," containing instructions for detecting and removing the new virus. That definition would be manually added to the set of definitions used by the antivirus program, and the new definitions placed on update servers for users of the antivirus program to download. Although some antivirus programs had begun to include scheduled update facilities that would automatically download and install the latest available definitions at some fixed interval, the rest of the process was almost entirely manual, and therefore limited in speed by the availability of skilled humans.

The immune system automates every stage of this process. The antivirus software on a protected client system uses a variety of heuristic methods to detect and identify files or other objects that may contain a new virus. A suspect file is encrypted and securely transmitted to an analysis center, where it is exercised and encouraged to spread within a protected environment. After it spreads, it is automatically analyzed, new detection and repair information is extracted, and the updated definitions are tested and provided to both the original infected system and to any other systems that are registered to receive automatic updates. At various stages of the process (especially during analysis) the system will defer to human experts if the virus does not yield to automatic methods, but for a significant percentage of new viruses the entire process proceeds without human intervention.

Various aspects of the immune system contain lessons for the development of secure autonomic elements. Perhaps the most important is that in a highly connected world, security threats can spread very quickly, and it is vital that the security response be correspondingly quick. Autonomic elements concerned with security must also be able to function under the sudden heavy loads often caused by widespread security incidents; the immune system uses a hierarchical network of gateways that cache the results of virus analysis and thus prevent a sudden flood of suspect files from bogging down the analysis center.

Autonomic elements must be able to function under the sudden heavy loads often caused by widespread security incidents.

Autonomic elements should also have the ability to ask for human confirmation of security-relevant actions. The component of the immune system that forwards a suspect file from a protected machine to the central analysis center can be configured to ask an administrator for confirmation before sending. At the same time, it should also be possible to configure the system so that this confirmation is not required; where a sufficient trust relationship exists between the protected enterprise and the owner of the analysis center, the confirmation can be turned off, so that suspect files are immediately forwarded. The security policies that govern autonomic elements must be flexible enough to allow this sort of configurability in connection with human confirmation.

Another lesson learned from the development and deployment of the antivirus immune system is a psychological one. While the system was being developed, considerable skepticism was expressed by experts in the field (including some of the antivirus experts involved in the development of the system) about the feasibility of creating an automatic system that would perform as well as a skilled human at critical security-related tasks (such as deriving recognition definitions for computer viruses). Those fears proved unfounded. Although the system does have decision points at which it can defer to a human expert when a particular virus is not automatically analyzable, for a typical virus the system does as well or better than human experts. In particular, the virus recognition patterns extracted by the immune system proved to be more powerful for detection, and less prone to false alarms, than those manually chosen by human experts.

The antivirus immune system we have just described is not the only computer security system inspired by biological analogies; see for instance Reference 15 for another approach inspired by biological immune systems and Reference 16 for a security system inspired by homeostatic processes.

Secure distributed storage. The central idea of the Secure Distributed Storage (SDS) solution is to efficiently spread important information over several separate servers in such a way that it is highly available, reliable, self-correcting, and self-protecting.[17] Let us assume that we want to store a file in an SDS system. The SDS system does not simply send a copy of the file to each server, since that would require every server to have enough storage to hold a complete copy of the file. Instead, the file is processed in a special way that breaks up the file into pieces, with each piece being significantly smaller than the original file. Each piece is then sent to a different server. This provides very efficient distributed storage.

Because of the special process that was used to break the file into pieces, one has only to retrieve the original pieces from just over half of the servers in order to completely reassemble the original file. It does not matter which pieces are retrieved, as long as at least half of them are. This allows for the other servers to have gone off-line or otherwise be unavailable. In addition, the SDS also provides a means by which compromised, malicious servers may be detected when a retrieval operation is underway. These capabilities allow the SDS to provide highly available, self-healing storage.

An SDS system consists, by definition, of several distinct servers. When depositing a file into an SDS system, the user can initiate the deposit of the file via any one of these servers. That is, any one of the available servers in an SDS system can act as the user's "gateway" for storing files. Similarly, when requesting a retrieval, the user may direct the request to any available server of the SDS. This also eliminates the need for the user's system to contact each of the distributed servers when attempting a deposit or retrieval. This transparency among the servers (auto-

nomic elements) directly supports more autonomic behavior.

Finally, an SDS system can be configured to be self-protecting. That is, since any of the servers might be down, disconnected from the network, or compromised by an attacker at any time, there must be some means for reestablishing the trust of that server. If a server was unavailable or disconnected, when it becomes available once again it sends the other servers a "What did I miss?" message and they all cooperate to bring that server up to date. Similarly, if a server has been compromised, that condition can be detected, and that server is automatically ignored until it can be reinitialized to restore its trustworthiness. In addition, the SDS system is capable of changing the representation of the data that it holds, so that information "captured" by an attacker who has broken into the system is useless. This is achieved by having the system periodically shuffle the pieces of a stored file across all the servers. This proactive security feature provides the self-protecting attribute of this autonomic system.

Conclusion

No functioning system is perfectly secure, and autonomic systems will be no exception. The development of autonomic systems cannot be delayed until we have found final solutions to all the corresponding security challenges, since such final solutions will never be available. On the other hand, for autonomic computing to succeed, it must be, and must be perceived as being, secure enough that its benefits outweigh the risks. In this paper, we have outlined a number of security and privacy challenges facing those designing and developing autonomic systems, and also a number of ways that autonomic principles can be used to make systems more secure than they are today.

There is a need for further research in many areas. Some of the key needs identified above include:

- Ways to represent and reason about the security and privacy policies that govern autonomic systems
- Ways to represent and reason about security states, and the trust relationships between elements
- Criteria and methods for effectively differentiating between normal system failures and failures caused by malicious attacks
- Policies and algorithms for making autonomic elements that are resistant to fraud and persuasion
- Common languages and taxonomies for communicating and negotiating about security and privacy states and policies
- Ways to construct individual autonomic elements so that their collective behavior is both trustworthy and trusted

Autonomic computing offers as least as many benefits in the security area as it does challenges. The complexity of modern computing systems makes secure systems administration a daunting task and one that is seldom done well in practice. Recent advances, including the growing use of automatic intrusion detection systems, secure embedded processors, proactive security measures, and automated virus response, have helped take some of the burden of security maintenance off overloaded system administrators, but there is much more to do. By making computing systems directly aware of the security policies that apply to them, and giving the systems the ability to conform their actions to those policies, the techniques of autonomic computing will help create systems that are increasingly and consistently secure.

**Trademark or registered trademark of Sun Microsystems, Inc.

Cited references

1. P. Horn, *Autonomic Computing: IBM's Perspective on the State of Information Technology*, IBM Corporation (October 15, 2001); available at http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.
2. C. C. Palmer, "Ethical Hacking," *IBM Systems Journal* **40**, No. 3, 769–780 (2001).
3. "The Dawning of the Autonomic Computing Era," by A. G. Ganek and T. A. Corbi, *IBM Systems Journal* **42**, No. 1, 5–18 (2003, this issue).
4. CERT® Advisory CA-2001-19: "Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL, CERT Coordination Center (2001); see http://www.cert.org/advisories/CA-2001-19.html.
5. D. Dittrich, "DDoS: Is There Really a Threat?," Usenix Security Symposium 2000; see http://staff.washington.edu/dittrich/talks/sec2000/.
6. P. Zimmerman, *PGP User's Guide*, MIT Press, Cambridge, MA (1994).
7. "Internet X.509 Public Key Infrastructure Certificate and CRL Profile," Internet standards track protocol document, RFC2459; see http://www.rfc-editor.org/rfc/rfc2459.txt.
8. United States Department of Defense, *Trusted Computer System Evaluation Criteria*, Technical Report DoD 5200.28-STD (1985).
9. L. Gong, *Inside Java 2 Platform Security*, Addison-Wesley Publishing Co., Boston, MA (1999).
10. M. Gerken, "Statistical-Based Intrusion Detection"; see http://www.sei.cmu.edu/str/descriptions/sbid.html (1997).
11. *Proceedings of the International Conference on Autonomous Agents*, ACM, Montreal, Canada (2001). See http://autonomousagents.org/.
12. "Special Issue on Deception, Fraud, and Trust in Agent So-

cieties," C. Castelfranchi et al. Editors, *Applied Artificial Intelligence Journal* **14**, No. 8 (2000).
13. World Wide Web Consortium, Platform for Privacy Preferences (P3P) Project (2002); see http://www.w3.org/P3P/.
14. S. R. White, M. Swimmer, E. Pring, W. Arnold, D. Chess, and J. F. Morar, "Anatomy of a Commercial-Grade Immune System," *Proceedings of the Ninth International Virus Bulletin Conference* (1999).
15. S. Hofmeyr and S. Forrest, "Architecture for an Artificial Immune System," *Evolutionary Computation* **7**, No. 1, 1289–1296 (2000).
16. A. Somayaji and S. Forrest, "Automated Response Using System-Call Delays," *Usenix Security Symposium* (2000).
17. J. Garay, R. Gennaro, C. Jutla, and T. Rabin, "Secure Distributed Storage and Retrieval," *Theoretical Computer Science* **243**, No. 1–2, 363–389 (2000).

Accepted for publication October 21, 2002.

David M. Chess *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: chess@us.ibm.com).* Mr. Chess joined IBM at the Thomas J. Watson Research Center in 1981. He has worked on mainframe performance management, computer conferencing, workstation technology, computer security, and virus prevention. He was on the team that developed and supported IBM AntiVirus, and he is currently a research staff member in the Massively Distributed Systems group, working on emergent security problems and security for active content and autonomic systems, as well as the architecture of autonomic computing elements. He has an A.B. degree in philosophy from Princeton University, and an M.S. degree in computer science from Pace University.

Charles C. Palmer *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: ccpalmer@us.ibm.com).* Dr. Palmer is senior manager of the Network Security and Cryptography departments at the Thomas J. Watson Research Center. His teams work in the areas of cryptography research, Internet security technologies, Java security, Web services, privacy, and in the Global Security Analysis lab (where they are known as the "ethical hackers"), which he cofounded in 1995. Dr. Palmer frequently speaks on the topics of computer and network security at conferences around the world. He received a Ph.D. degree from Polytechnic University in computer science in 1994, where he was also an adjunct professor of computer science from 1993 to 1997. He holds five patents and has several publications from his work at IBM and Polytechnic.

Steve R. White *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: srwhite@us.ibm.com).* Dr. White received a Ph.D. from the University of California at San Diego in theoretical physics in 1982. He accepted a postdoctoral fellowship at IBM Research, where he later became a research staff member. He has published in the fields of condensed matter physics, optimization by simulated annealing, software protection, computer security, computer viruses, and information economies. He holds roughly a dozen patents in related fields. He was elected to the IBM Academy, which advises the company on technology policy and direction, and has received IBM's highest technical award for his work. At the IBM Thomas J. Watson Research Center, he is currently senior manager of the Autonomic Computing group, which explores how we can build computing systems that have billions of components and are nevertheless self-managing.