In *Self-Adaptive Software*, P. Robertson, H. Shrobe, and R. Laddaga, editors, Berlin: Springer Verlag, 2001, pp. 40-49 (revised papers from the First International Workshop on Self-Adaptive Software (IWSAS 2000)).

Active Trust Management for Autonomous Adaptive Survivable Systems (ATM's for AAss's)

Howard Shrobe and Jon Doyle
MIT AI Lab, Cambridge, MA 02139, USA,
hes@ai.mit.edu,
WWW home page: http://www.ai.mit.edu/people/hes/

Abstract

The traditional approaches to building survivable systems assume a framework of absolute trust requiring a provably impenetrable and incorruptible Trusted Computing Base (TCB). Unfortunately, we don't have TCB's, and experience suggests that we never will.

We must instead concentrate on software systems that can provide useful services even when computational resource are compromised. Such a system will 1) Estimate the degree to which a computational resources may be *trusted* using models of possible *compromises*. 2) Recognize that a resource is compromised by relying on a system for long term *monitoring and analysis* of the computational infrastructure. 3) Engage in *self-monitoring, diagnosis and adaptation* to best achieve its purposes within the available infrastructure. All this, in turn, depends on the ability of the application, monitoring, and control systems to engage in *rational decision making* about what resources they should use in order to achieve the best ratio of expected benefit to risk.

1 A Scenario

Within the MIT Artificial Intelligence Laboratory an ensemble of computers runs a Visual Surveillance and Monitoring application. On January 12, 2001 several of the machines experience unusual traffic from outside the lab. Intrusion Detection systems report that several password scans were observed. Fortunately, after about 3 days of varying levels of such activity, things seem to return to normal; for another 3 weeks no unusual activity is noticed. However, at that time, one of the machines (named Harding) which is crucial to the application begins to experience unusually high load averages and the application components which run on this machine begin to receive less than the expected quality of service. The load average, degradation of service, the consumption of disk space and the amount of traffic to and from unknown outside machines continue to increase to annoying levels. Then they level off. On March 2, a second machine in the ensemble (Grant) crashes; fortunately, the application has been written in a way which allows it to adapt to unusual circumstances. The system considers whether it should migrate the computations which would normally have run on Grant to Harding; however, these computations are critical to the application. The system decides that in spite of the odd circumstances noticed on Harding earlier, it is a reasonable choice.

Did the system make a good choice? It turns out it did. The system needed to run those computations somewhere; even though Harding was loaded more heavily than expected, it still represented the best pool of available computational resources, other machines were even more heavily loaded with other critical computations of the application. But what about all the unusual activity that had been noticed on Harding? It turns out that what had, in fact, transpired is that hackers had gained access to Harding by correctly guessing a password; using this they had set up a public FTP site containing among other things pirated software and erotic imagery. They had not, in fact, gained root access. There was, therefore, no worry that the critical computations migrated to Harding would experience any further compromise. (Note: the adaptive system in this story is fictional, the compromised computers reflect an amalgam of several real incidents).

Let's suppose instead that (1) the application was being run to protect a US embassy in Africa during a period of international tension (2) that we had observed a variety of information attacks being aimed at Harding earlier on (3) that at least some of these attacks are of a type known to be occasionally effective in gaining root access to a machine like Harding and that (4) they are followed by a period of no anomalous behavior other than a periodic low volume communication with an unknown outside host. When Grant crashes, should Harding be used as the backup? In this case, the answer might well be the opposite; for it is quite possible that an intruder has gained root access to Harding; it is also possible that the intent of the intrusion is malicious and political. It is less likely, but still possible, that the periodic communication with the unknown outside host is an attempt to contact an outside control source for a "go signal" that will initiate serious spoofing of the application. Under these circumstance, it is wiser to shift the computations to a different machine in the ensemble even though it is considerably more overloaded than Harding.

What can we learn from these examples?

1. It is crucial to estimate to what degree and for what purposes a computer (or other computational resource) may be *trusted*, as this influences decisions about what tasks should be assigned to them, what contingencies should be provided for, and how much effort to spend watching over them.

2. Making this estimate depends in turn on having a model of the possible ways in which a computational resource may be *compromised*.

3. This in turn depends on having in place a system for long term *monitoring and analysis* of the computational infrastructure which can detect patterns of activity such as "a period of attacks followed by quiescence followed by increasing degradation of service". Such a system must be capable of assimilating information from a variety of sources including both self-checking observation points within the application itself and intrusion detection systems.

4. The application itself must be capable of *self-monitoring and diagnosis* and capable of *adaptation* so that it can best achieve its purposes with the available infrastructure.

5. This, in turn, depends on the ability of the application, monitoring, and control systems to engage in *rational decision making* about what resources they should use in order to achieve the best ratio of expected benefit to risk.

Systems that can do the above things can be resilient in the face of concerted information attacks. They can carry on through non-malicious intrusions; that is they can figure out when compromises within the infrastructure can't actually hurt them.

Our claim is simple but revolutionary: "Survivable systems make careful judgments about the trustworthiness of their computational environment and make rational resource allocation decisions accordingly."

The claim is deceptively simple: To make it real one needs to develop serious representations of the types of compromises, of the trustworthiness of a resource, and of the goals and purposes of the computational modules within an application. One also needs to build monitoring, analysis and trend detection tools and adaptive computational architectures. Finally, one needs to find a way to make the required rational decision making computationally tractable. None of this is easy, but we have ideas and ongoing projects addressing each of these issues.

2 Trust in Survivable Systems

Traditional approaches to building survivable systems assume a framework of absolute trust. In this view, survivable systems require a provably impenetrable and incorruptible Trusted Computing Base (TCB). Unfortunately, we don't have TCB's, and experience suggests that we never will.

Instead, we will need to develop systems that can survive in an imperfect environment in which any resource may have been compromised to some extent. We believe that such systems can be built by restructuring the ways in which systems organize and perform computations. The central thrust of this approach is a radically different viewpoint of the trust relationships that a software system must bear to the computational resources it needs.

The traditional TCB-based approach takes a binary view of trust; computational resources either merit trust or not, and non-trusted resources should not be used. The traditional view also considers trustworthiness as a nearly static property of a resource: trust lost is never regained, short of major system reconstruction. Consequently, these systems wire decisions about how and where to perform computations into the code, making these decisions difficult to understand, and preventing the system from adapting to a changing runtime environment.

We agree with this viewpoint on the crucial role of the assessment and management of trust, but reject the assumptions about the binary, static nature of trust relationships as poor approximations to real-life computing situations. We instead base our approach on a different, more realistic set of assumptions:

1. All computational resources must be considered suspect to some degree, but the degree of trust that should be accorded to a computational resource is not static, absolute, or known with full certainty. In particular, the degree of trustworthiness

may change with further compromises or efforts at amelioration in ways that can only be estimated on the basis of continuing experience. The system must thus continuously and actively monitor the computational environment at runtime to gather evidence about trustworthiness and to update its trust assessments.

2. Exploiting assessments of trustworthiness requires structuring computations into layers of abstract services, with many distinct instantiations of each service. These specific instantiations of a service may vary in terms of the fidelity of the answers that they provide, the conditions under which they are appropriate, and the computational resources they require. But since the resources required by each possible instantiation have varying degrees of trustworthiness, each different way of rendering the service also has a specific risk associated with it.

3. The best method for exploiting assessments of trustworthiness requires making explicit the information underlying decisions about how (and where) to perform a computation, and on formalizing this information and the method used to make the decision in a decision-theoretic framework. The overall system adapts to the dynamism of the environment and to the changing degrees of compromise in its components by deciding dynamically which approach to rendering a service provides the best likelihood of achieving the greatest benefit for the smallest risk. We do not require that the system uses explicit decision-theoretic calculations of maximal expected utility to make runtime decisions; the system may instead use the decision-theoretic formalizations to decide on policies and policy changes, which then are used to compile new code governing the relevant behaviors.

4. The system must consider selected components to be fallible, even if it currently regards them as trustworthy, and must monitor its own and component behaviors to assure that the goals of computations are reached. In the event of a breakdown, the system must first update its assessments of the trustworthiness of the computational resources employed and then select an alternative approach to achieving the goal.

2.1 How Active Trust Management can support Autonomous Adaptive Survivable Systems

These considerations motivate an architecture both for the overall computational environment (Active Trust Management) and for the application systems which run within it (Autonomous Adaptive Survivable Systems). The environment as a whole must constantly collect and analyze data from a broad variety of sources, including the application systems, intrusion detection systems, system logs, network traffic analyzers, etc. The results of these analyses inform a "Trust Model", a probabilistic representation of the trustworthiness of each computational resource in the environment. The application systems use this trust model to help decide which resources should be used to perform each major computational step; in particular, they try to choose that resource which will maximize the ratio of expected benefit to risk. This "rational decision making" facility is provided as a standard utility within the environment. The application systems also

monitor the execution of their own major components, checking that expected post-conditions are achieved. If these conditions fail to hold, diagnostic services are invoked to determine the most likely cause of the failures and thereby to determine the most promising way to recover. In addition to localizing the failure, the diagnostic services can also infer that underlying elements of the computational infrastructure are likely to have been compromised and these deductions are forwarded to the monitoring and analysis components of the environment to help inform its assessments of trustworthiness. Finally, having accumulated sufficient evidence, the monitoring and analysis systems may decide that it is likely that some resource has, in fact, been compromised. This will have an immediate impact if the resource is being used to perform a computation which would be damaged by that specific form of compromise; in such cases, the monitoring and analysis components transmit "alarms" into the running application, causing it to abandon its work and to immediately initiate recovery efforts.

Thus the application system forms a tight feedback control loop whose goal is to guarantee the best possible progress towards providing the services the application is intended to provide to its users (i.e. the applications are Autonomous Adaptive Survivable Systems "AASS's"). The computational infrastructure also forms a feedback control loop whose goal is to maintain an accurate assessment of the trustworthiness of the computational resources; this assessment can then inform the application systems' decision making and self-monitoring which in turn helps inform the long-term assessments of trustworthiness (Active Trust Management "ATM").

This vision leads us to focus our efforts in four major areas:

1. Models of Trust and Compromise
2. Perpetual Analytic Monitoring
3. Autonomous Adaptive Survivable Systems
4. Rational Decision Making in a Trust-Driven Environment

3 Models of Trust and Compromise

Making rational decisions about how to use resources in an environment of imperfect trust requires information about what resources can be trusted, and for what purposes. We are developing models of trust states that go beyond mere information about whether or how a system has been subject to attack to represent whether or how different properties of the system have been compromised, and finally to represent whether they can be trusted for a particular purpose even if compromised. We also represent the degree to which these judgments should be suspected or monitored.

These models provide the point of intersection among all the other elements of the proposed approach. Trust plays a central role in resource allocation decisions. All decisions about what to do must be based on beliefs about the situation in which the action is to be taken. We can think of the degree of trust one places in a system as the degree to which one is willing to rely on the proper functioning of the system without also dedicating unusual effort to preparing for the contingency of failure. Since preparations

for contingencies consume resources, this makes trust management a central resource allocation issue.

The trust model is organized into three levels above that of raw behavior:

The lowest level of the trust model represents the results of initial interpretations such as *attacks* and *anomalous behavior*. At this level we collect, filter and organize the necessary information so that it can trigger trend templates and feed into Bayesian inference networks. As we saw in our scenarios, we are not primarily interested in what attacks or anomalous behaviors have taken place, but rather in what they imply about what compromises might actually be present.

The middle level of the trust model deals with *compromises*. The attack level only tells us that malicious or anomalous activity has taken place. But what we are interested in is whether someone has actually succeeded in an attack and has used that to exploit or corrupt resources. That such a compromise has occurred can be inferred by matching the temporal patterns of activity to a template for a particular compromise. In the scenario we saw an example of this in which the gaining of unauthorized user level access was indicated by the temporal pattern of password sweeps followed by quiescence followed by increasing resource consumption.

The highest level of the trust model deals with *trustworthiness*. The fact that a resource has been compromised does not in and of itself imply that it is totally unsafe to utilize it. That conclusion depends on the precise way in which the consumer wants to utilize the resource as well as on assessments of the *intention* of the compromiser. In our scenarios, we presented two different examples: in the first the system was compromised by "teenaged hackers" looking for free resources, in the second it was compromised by state-sponsored malicious agents. Clearly, we should generally be more wary of using a resource in the second case than the first; but if we are not very sensitive to quality of service and perhaps only care about the integrity of our data, then the first case is not all that risky.

Knowledge of attack types guides the organization's attempts to defend against future attacks. Knowledge of compromises indicates the threats to operations. Knowledge of trust states guides how the organization carries on in the face of partially-understood compromises. Because intent plays a central role, it too must be modeled throughout the three layers, moving from raw reports about behavior at the base level, to statements about intent in the middle layer and finally entering into assessments of trustworthiness at the highest level.

4 Perpetual Analytic Monitoring keeps the Trust Model current by detecting Trend Patterns which are indicative of compromise

The Perpetual Analytic Monitoring component of our project is based onthe MAITA system which consists of a library of monitoring methods, an architecture for operating networks of monitoring processes, and a flexible, display-oriented control system for quickly constructing, composing, modifying, inspecting, and controlling monitoring networks [3, 2, 1].

The goal of Perpetual Analytic Monitoring is to assess the trustworthiness of the computational resources in the environment. The scenario illustrates that building a trust model involves more than just detecting an intrusion. Indeed, what was important was a template of activity patterns consisting of several temporal regions: First there was a period of attacks (particularly password scans). Then there was a "quiescent period". Then there was a period of increasing degradation of service. Finally, there was a leveling off of the degradation but at the existing high level. We call such a temporal pattern a "trend template".

A trend template has a temporal component and a value component. The temporal component includes landmark time points and intervals. Landmark points represent significant events in the lifetime of the monitored process. They may be uncertain in time, and so are represented with time ranges. Intervals represent periods of the process that during which constant dynamics obtain. Value changes are described in terms of various standard curves. The value component characterizes constraints on individual data values and specifies constraints that must hold among different data streams. The representation is supported by a temporal utility package (TUP) that propagates temporal bound inferences among related points and intervals [12, 11].

The MAITA monitoring method library includes entries at varying levels of computational detail. For example, the most abstract levels speaks of constructing and comparing a set of hypotheses about what is going on, without providing any details about how the hypotheses are constructed or compared. The intermediate level, uses the TrenD_x [8, 7, 13, 10, 6] trend monitoring system to recognize trend templates.

Trend templates are necessary, but not sufficient in themselves. We also need to make inferences about the factual situation at hand (e.g., are international tensions rising?) and about the intentions, and states of mind of significant players (e.g., would it be likely that they are trying to attack me?). All of these inferences involve the combining of evidence to provide assessments of the likelihood of certain propositions. Bayesian networks provide a convenient formalism for representing and reasoning with basic probabilistic information.

The principal goal of our Monitoring and Analysis tools is to keep the Trust Model current. However, when these tools have achieved a high degree of confidence that a compromise has occurred, the monitoring and analysis system must generate an alarm asking currently executing application components to rollback and attempt to use alternative strategies and resources.

5 Autonomous Adaptive Survivable Systems use Trust Models and and Self Reflection to select computational strategy and to recover from compromise

Autonomous Adaptive Survivable Systems have the goal of adapting to the variations in their environment so as to render useful services under all conditions. In the context of Information Survivability, this means that useful services must be provided even when there have been successful information attacks.

AASS's achieve adaptivity in two ways: First, they include many alternative implementations of the major computational steps, each of which achieves the same goal but in different ways. An AASS is therefore a Domain Architecture; it capitalizes on the "variability within commonality" which is typical of the service layers in any software domain. Each service is annotated with specifications and is provided with multiple instantiations optimized for different purposes.

AASS's are implemented in a *Dynamic* Domain Architecture: all the alternative instantiations of each service, plus the annotations describing them are present in the run-time environment; the decision of which alternative instantiation of a service to employ is made dynamically and as late as necessary. Before each step is actually initiated, the system first assesses which of these is most appropriate in light of what the trust model tells it about compromises and trustability. We may view this as an extremely dynamic and information rich version of Object-Oriented Programming in which method invocation is done in decision-theoretic terms, i.e., we invoke that method most likely to succeed given the current trust state.

The second way in which AASS's achieve adaptivity is by noticing when a component fails to achieve the conditions relied on by other modules, initiating diagnostic, rollback and recovery services. This depends on effective monitoring of system performance and trustworthiness which in turn requires a structured view of the system as decomposed into modules, together with teleological annotations that identify prerequisites, post-conditions and invariant conditions of the modules. These teleological models also include links describing how the post-conditions of the modules interact to achieve the goals of the main system and the prerequisites of modules further downstream. Tools in the development environment use these representations to generate run-time monitors that invoke error-handling services if the conditions fail to be true. The exception-management service is informed by the Dynamic Domain Architecture's models of the executing software system and by a catalog of breakdown conditions and their repairs; using these it diagnoses the breakdown, determines an appropriate scope of repair; possibly selects an alternative to that invoked already and then restarts the computation.

Thus, we remove exception handling from the purview of the programmer, instead treating the management of exceptional conditions as a special service provided by cooperating services in both the run-time and development environments. Model-based diagnostic services [4, 9, 5, 14] play a key role in an AASS's ability to recover from a failure. This is done by providing models not only of the intended behavior of a component but also of its likely failure modes. These application-layer models are linked to models of the behavior of the computational infrastructure on which the application components execute. Again these include models both of expected and compromised behavior. The diagnostic task then is to identify the most likely set of such models which is consistent with the observed behavior. This helps the application decide how to recover from the failure and restore normal functioning. It also provides evidence to the overall monitoring environment about the trustworthiness of the underlying computational resources, particularly when the most likely diagnosis is that one of the resources has been compromised.

In summary, a dynamic domain architecture provides the following services that

enable Autonomy, Adaptivity and Survivability in the applicatino System:

1. Synthesis of code that selects which variant of an abstract operator is appropriate in light of run-time conditions.

2. Synthesis of monitors that check whether conditions expected to be true at various points in the execution of a computation are in fact true.

3. Diagnosis and isolation services that locate the cause of an exceptional condition, and characterize the form of the breakdown which has transpired.

4. Selection of alternative methods that achieve the goal of the failed computation using different means (either by trying repairs or by trying alternative implementations, or both).

5. Rollback and recovery services that establish a consistent state of the computation from which to attempt the alternative strategy.

6. Reallocation and re-optimization of resource allocations in light of the resources remaining after the breakdown and the priorities obtaining at that point. These services may optimize the system in a new way in light of the new allocations and priorities.

6 Rational Decision Making uses decision-theoretic models and the Trust Model to control decisions about component selection and resource allocation

We assess system trustworthiness and performance according to the trust and teleological models in order to make decisions about how to allocate computational resources. To ensure that these decisions represent a good basis for system operation, we are developing detailed decision-theoretic models of trustworthiness, suspicion, and related concepts as applied to information systems and their components. These models will relate notions such as attractiveness of a system as a target, likelihood of being attacked, likelihood of being compromised by an attack, riskiness of use of the system, importance or criticality of the system for different purposes, etc.

The models will also relate estimates of system properties to an adaptive system of decision-theoretic preferences that express the values guiding the operation of both system modules and the system as a whole. We are develop mechanisms that use these elements to allocate system resources optimally given task demands, trustworthiness judgments, and the resources available.

We believe that the combination of these ideas will lead to systems that can adapt to and survive infomation attacks.

References

[1] C. Cao, J. Doyle, I. Kohane, W. Long, and P. Szolovits. The MAITA monitoring library and language. In preparation, 1998.

[2] C. Cao, J. Doyle, I. Kohane, W. Long, and P. Szolovits. The MAITA monitoring network architecture. In preparation, 1998.

[3] C. Cao, J. Doyle, I. Kohane, W. Long, and P. Szolovits. The MAITA system: an overview. In preparation, 1998.

[4] R. Davis, H. Shrobe, W. Hamscher, K. Wieckert, M Shirley, S. Polit
Diagnosis Based on Descriptions of Structure and Function
AAAI, National Conference on Artificial Intelligence Pittsburgh, PA., 1992 pp 137-142.

[5] J. deKleer and B. Williams
Reasoning About Multiple Faults
AAAI, National Conference on Artificial Intelligence, Philadelphia, Pa., 1986, pp 132-139.

[6] J. Fackler, I. J. Haimowitz, and I. S. Kohane. Knowledge-based data display using trendx. In *AAAI Spring Symposium: Interpreting Clinical Data*, Palo Alto, 1994. AAAI Press.

[7] I. J. Haimowitz and I. S. Kohane. Automated trend detection with alternate temporal hypotheses. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 146–151, Chambery, France, 1993.

[8] I. J. Haimowitz and I. S. Kohane. An epistemology for clinically significant trends. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 176–181, Washington, DC, 1993.

[9] W. Hamscher. Modeling digital circuits for troubleshooting. *Artificial Intelligence*, 51:223–227, 1991.

[10] I. Kohane and I. Haimowitz. Hypothesis-driven data abstraction. In *Symposium on Computer Applications in Medical Care*, Washington, DC, 1993.

[11] I. S. Kohane. Temporal reasoning in medical expert systems. In R. Salamon, B. Blum, and M. Jørgensen, editors, *MEDINFO 86: Proceedings of the Fifth Conference on Medical Informatics*, pages 170–174, Washington, Oct. 1986. North-Holland.

[12] I. S. Kohane. Temporal reasoning in medical expert systems. TR 389, Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, Apr. 1987.

[13] I. S. Kohane and I. J. Haimowitz. Encoding patterns of growth to automate detection and diagnosis of abnormal growth patterns. *Pediatric Research*, 33:119A, 1993.

[14] B. Williams and J. deKleer. Diagnosis with Behavior Modes. In *Proceedings of the 11th Joint Conference on Artificial Intelligence, IJCAI-89, pages 1324–1330*, Detroit MI, 1989 .