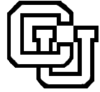

Department of Computer Science
University of Colorado
Boulder, CO 80309-0430

# Intrusion Management Using Configurable Architecture Models

Dennis Heimbigner and Alexander Wolf
{dennis, alw}@cs.colorado.edu

Technical Report 929-02

April 5, 2002

Department of Computer Science
University of Colorado
Boulder, Colorado 80309-0430

# Intrusion Management Using Configurable Architecture Models

## New Security Paradigms Workshop Position Paper

Dennis Heimbigner and Alexander Wolf
Department of Computer Science
University of Colorado
{dennis, alw}@cs.colorado.edu

## 1 Introduction

Software is increasingly being constructed using the component-based paradigm in which software systems are assembled using components from multiple sources. Moreover, these systems are increasingly dynamic; a core set of components is assembled and then new functionality is provided as needed by dynamically inserting additional components.

A newer trend closely associated with the use of component-based software is the *post-development* use of configurable run-time architecture models describing the structure of the software system. These models are coming out of the software engineering community and are being used to manage component-based systems at deployment and operations time. The key aspect of this trend is that these models accompany the software system and provide the basis for defining and executing *run-time* monitoring and reconfiguration of these systems.

We believe that these models have the potential for providing a new and important source of information that can be exploited to improve the management of intrusions directed against these software systems. Our hypothesis is that they can provide a common framework for integrating and managing all phases of intrusion defenses: phases including intrusion detection, response, and analysis. We will show how these models can provide a framework around which to organize intrusion-related data. We will also show how architecture-driven reconfiguration can provide improved response, and how inconsistencies between the models and the actual system state can support application-level anomaly detection and computer forensics analysis.

Our approach directly challenges the existing practice of basing intrusion management on low-level features such as network packets or system call traces. The former is too far removed from the operation of application software, and the latter provides at best limited insight into the operation of the application. We recognize that these low-level features have been used because they are readily available. However, this has resulted in treating applications as monolithic black boxes whose internal operation is considered "out of scope". This must change because most intrusions now appear to exploit application level vulnerabilities: email viruses and buffer overflows, for example. We are saying that a new class of information is becoming available and it should be used to improve intrusion management across the board.

In the remainder of this paper, we will first provide some background for our approach. We will discuss the meaning of intrusion management and our intrusion management life cycle. Then in subsequent sections we will examine the application of configurable architecture information to improving the operation of intrusion management tools. Finally, we will discuss some research issues that must be solved to achieve effective use of architecture information in intrusion management.

## 2 Background

The idea of applying configurable architecture models to intrusion management comes from an analysis of our DARPA funded Willow project at the University of Colorado. Our goal was to investigate the extension of Willow to additional areas of intrusion management. Willow already demonstrates the application of configurable run-time architecture models to intrusion prevention and repair, and this was enabled, in turn, by the recent availability of powerful architecture models and support for the dynamic configuration of software systems.

### 2.1 Willow

The Willow project [10] represents an early example of a framework that can utilize architecture information to improve intrusion management. Willow provides a secure, automated framework for reconfiguration of large-scale, heterogeneous, distributed systems. Reconfiguration is used to tolerate intrusions and faults yet still continue to provide an acceptable level of service. Willow supports both *proactive* reconfiguration and *reactive* reconfiguration (repair) of software systems.

Proactive reconfiguration adds, removes, and replaces components and interconnections to cause a system to assume *postures* that achieve specific intrusion tolerance goals, such as increased resilience to specific kinds of attacks or increased preparedness for recovery from specific kinds of failures. The term "posture" refers to a set of policies and procedures ensuring survivability against a particular set of threats to service while taking into account the tradeoff of performance against protection. In Willow, a posture is embodied as a particular configuration of components providing a particular level of functionality within a particular range of performance and security parameters. Proactive reconfiguration can also cause a relaxation of tolerance procedures once a threat has passed, in order to reduce costs, increase system performance, or even restore previously excised data and functionality.

In a complementary fashion, reactive reconfiguration adds, removes, and replaces components and interconnections to restore the integrity of a system in bounded time once an intrusion has been detected and the system is known or suspected to have been compromised. Recovery strategies made possible by reactive reconfiguration include restoring the system to some previously consistent state, adapting the system to some alternative non-compromised configuration, or gracefully shedding non-trustworthy data and functionality.

Willow currently focuses on the problem of responding to intrusions. This is a problem that is still largely handled by manual processes. Willow replaces this with automated reconfiguration responses that can react to disruptions with a speed, accuracy, and scale not achievable by manual procedures.

### 2.2 Configurable Run-time Architecture Models

The field of software architecture [16] has evolved to meet the need for design notations for specifying structural and behavioral properties of complex systems constructed from coarse-grain building blocks. It was originally developed to be used at design time to support early analysis of software systems for high-level properties. It is only more recently that it has been adapted to operate at run-time.

Software architecture models provide high-level representations of the structure, behavior, and key properties of a software system. Such models involve (1) descriptions of the elements from which a system is built, (2) interactions among those elements, (3) patterns that guide their composition, and (4) constraints on these patterns. In general, a particular system is defined in terms of a collection of components, their interfaces, their interconnections (configuration), and interactions among them (connectors).

As architecture models moved from design time to run-time, it became important to be able to define multiple configurations for architectures. This is because a dynamic system requires the architecture model to also support *configurability*: the ability to modify the structure of a system to place it into a variety of *configurations*. Again, Willow provides a good example of this. The approach adopted in Willow is derived from our earlier Software Dock project [8]. An architecture model is annotated to represent a *system family*, which represents the possible *versions* and *variants* of the architecture of the system.

As is traditional in the configuration literature, a sequence of versions represents revisions of the system architecture over time. The set of variants represents the range of alternative architectures. Configuring a system is the process of choosing a specific family instance and modifying the run-time structure of a system to conform to the chosen instance. This typically involves the addition, deletion and modification of the structural elements represented by the architecture model: components, interfaces, connectors, and constraints. The specification of families in Willow is controlled by a set of properties that determine that instance. More information is provided in a previous paper [9].

We are deliberately using the term "architecture" somewhat broadly. In fact, we have identified a range of architecture sub-models that represent the structure of a software system at various stages of its operation from initial deployment to execution time. Section 5.1 and Appendix A discuss these sub-models in more detail.

Finally, we also are using the term "model" somewhat ambiguously. On one hand, we are using the term to mean a notation for representing many specific models of specific software systems. On the other hand, we use the term to refer to any specific instantiation representing a specific software system. We trust that context is sufficient to disambiguate the two meanings.

## 3 Intrusion Management Life Cycle

In order to organize our discussion showing how configurable run-time architecture models can benefit intrusion management, we will introduce a simple life cycle (Figure 1) that contains the primary activities for handling intrusions from the initial attack to the final prevention of repeat occurrences (if that is possible). It is important to note that this life cycle is defined from the *defender's* point of view. The goal of this life cycle is to provide a theme for integrating and relating

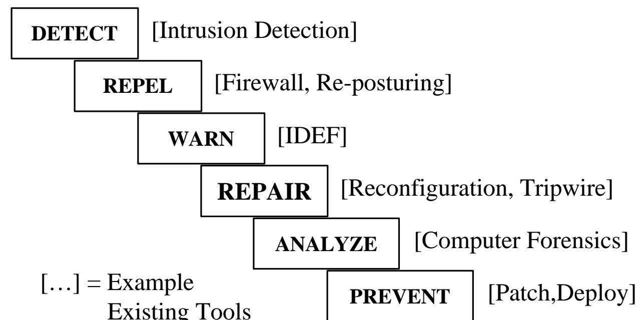


**Figure 1. Intrusion Management Life Cycle**

existing intrusion management components. It should not be confused with an *attack model*, which in its usual form describes the sequence of steps taken by an attacker to exploit a given vulnerability in order to gain special privileges in the defender's domain.

As illustrated in Figure 1, our life cycle consists of six phases.

1. Detect – The initial detection that an intrusion has commenced or is about to commence.

2. Repel – An early response that is intended to quickly blunt an attack.

3. Warn – Signals sent to other intrusion tools and to related software systems to warn of the presence of an intrusion and any details about its operation.

4. Repair – State changes necessary to recover some level of functionality and to undo damage after an intrusion has run its course.

5. Analyze – An examination of the system state after an intrusion to determine the nature of the attack and possible future defenses against it.

6. Prevent – Modifications to the system so as to make it resistant to similar intrusions in the future.

It is important to note that this life cycle is idealized – a feature it has in common with most life cycle diagrams. The steps will not always occur in strict sequence. In practice, many of these steps will occur in parallel or in arbitrary order: Repel and Warn, for example. Further, multiple attacks may occur simultaneously, so the life cycle will be multiply instantiated. Finally, there are implicit backward loops in the cycle to indicate that some steps may fail and need to be retried or that the output of one step can modify the actions of another. With those caveats, we will examine each phase and discuss the way in which architectural models can significantly improve the operation of that phase.

# 4 Applications of Models in Intrusion Management

Configurable run-time architecture models enable several capabilities applicable to intrusion management. The most important one is the capability to dynamically reconfigure systems (if they are designed for it - see Section 5.3). Monitoring the internal operation of a software system is another important capability. This can expose previously hidden information for use by external agents. Note that the two abilities are closely connected: monitoring may be enabled by dynamically inserting various probes into the software. A third capability involves checking the consistency of the architecture model, especially of a running system, against a snapshot of the current state of a system. Differences can provide clues about potential problems such as Trojan horses. All of these capabilities have roles in the intrusion management life cycle.

## 4.1 Detecting Intrusions

The initial phase of intrusion management is to detect that an intrusion is occurring or to detect warning signs of an impending attack. An intrusion detection system (IDS) is usually used for this purpose, and IDSs are the most common tool available for intrusion management.

Intrusion detection systems are traditionally classified as either signature based (looking for attacks) or anomaly-based (looking for deviations from good behavior) or specification-based (looking for deviations from a specification of good behavior). Existing signature and anomaly based intrusion detectors are typically targeted at a low-level event stream such as IP packets. Specification-based detectors raise the level somewhat by addressing system calls or log entries issued from an application program. This low-level focus has been reasonable because the event streams (packets and system calls and logs) are readily accessible.

We believe that combining component-based programs and architecture information can allow new forms of IDS that have substantially better insight into the internal operation of software systems. That is, it will no longer be necessary to treat the software as a black box. An ability to monitor the internal operation of a software system can be exploited to look for anomalous behavior. Thus, we can augment and complement any existing IDS to allow it access to more information to analyze in order to detect intrusions.

Architecture information can also support a secondary, but interesting, new capability: reconfiguration of the IDS. In our role in the ARO-sponsored Hi-DRA intrusion detection project, the IDS itself is treated as a complex distributed system subject to reconfiguration to include new sensors and to adapt to new classes of attacks.

### 4.2 Repelling Intrusions

A quick response to a detected intrusion may blunt, or at least soften, the effect of an attack before it has a chance to do real damage. At the moment, most response mechanisms are ad-hoc, often manual, and involve such things as changing the filter rules on a firewall.

Reconfiguration, as provided by Willow, has the potential for providing a more comprehensive mechanism for repelling intrusions. From the perspective of reconfigurable architectures, changing firewall rules is just a specific form of posturing in which the firewall is reconfigured dynamically to resist and attack. This approach can be generalized to support reconfiguration of other software systems as well, including the IDS, and application services such as web servers. To the extent that effective postures can be defined, reconfiguration then provides a common mechanism for implementing those postures.

### 4.3 Warning Others

Intrusion management should be a group activity where attacks against one software system or against one administrative domain generate warning messages to interested parties. The IETF is working on the Intrusion Detection Exchange Format (IDEF) [5] for encoding intrusion events. This is a follow-on to the DARPA Common Intrusion Detection Framework (CIDF) [19] effort.

Architecture information again allows the black box of software systems to be opened up to provide more information. Thus, it supports more detailed reporting of intrusion events against those systems. It is also possible to report the success or failure of architecture-based repel or repair attempts.

## 4.4 Repairing Damage

A successful attack can be expected to damage various components of a system. This includes defacing web pages, inserting Trojan horses or viruses, and deactivation of software systems (including intrusion management systems). To date, damage repair is mostly a slow, manual process involving the attention of a highly skilled system administrator. Some tools such as Tripwire [21] can significantly help in the restoration process.

Architecture-based tools such as Willow have the potential to automate much more of the repair process by using the expected run-time architecture as the specification against which the damaged state is compared. The difference between model and reality can be used to automatically reconstruct a working system by reconfiguring various software systems to clear out damaged components and to restart correct versions. Of course, it is important not to minimize the difficulties. Determining the state of the compromised system can be difficult, and repairing state in general is hard and requires careful checkpointing. Additionally, finding trusted sources from which to get undamaged components must also be addressed. Nevertheless, reconfigurable architecture models would appear to provide much of the information necessary to accomplish these tasks.

## 4.5 Analysis of Intrusions (Computer Forensics)

Computer forensics is the process of analyzing the state of a computer system after an apparent intrusion in order to determine the mechanisms and damage attributable to the intrusion. Simplifying somewhat, the state of the art in forensics involves examining raw data (i.e., core dumps and file system dumps) looking for anomalies at the level of file descriptors or file checksums. The Coroner's Toolkit [4] provides a set of tools to help with these examinations, but the process is still slow and labor intensive.

Architecture models again have the potential to provide more and better information to the analyst to improve his effectiveness. Automated tools that analyze core dumps can do this by matching the dumped state against the model and detecting and noting differences. Thus, the analyst is given an architecture-level view of what systems were running at the time of the failure and how those systems differed from the expected structures. The model also provides a general structure on which to hang additional annotations about other information such as file descriptors and process identifiers. Thus it can provide additional automated help in organizing all of the raw information extracted from the core dump.

## 4.6 Preventing Repeated Intrusions

The last stage in our intrusion management life cycle is prevention of future attacks. Of course this requires some knowledge about the nature of the attack, which must come from an analysis of the attack. The output of an analysis is some form of proactive response that can be applied to existing systems to place them into postures capable of resisting the attack in the future. Traditionally, this involves file patches. But patching of complex distributed systems, especially while they are running, can be difficult. For systems that cannot easily be stopped, the ability to reconfigure dynamically, using architecture models, provides an important new capability for responding quickly to intrusions.

# 5 Research Issues

We have outlined the application of configurable run-time architectures to intrusion management and have indicated how it might improve the current state of the art in each phase. Substantial research remains, however, in order to actually achieve these improvements. In the following sections we discuss some of the research topics that need addressing.

## 5.1 Architecture Sub-Models

We recognize that the "architecture" of a software system differs as the system moves through its operational life cycle. Thus, before being deployed, the architecture may refer to the whole family of deployable variants of the system depending on environmental details such as the target host operating system. At the time that the system is executed, the running system will have a structure that is dependent again on various environmental parameters, although it should be consistent with the deployed architecture. Appendix A expands on the kinds of sub-models we have identified as necessary.

## 5.2 Populating the Models

Given a set of modeling notations, we must construct models for specific software systems (and environments). We believe strongly in reusing existing information, so we have identified a set of sources for various kinds of information that can be used to construct specific model instance. Example sources include the DMTF CIM model (dmtf.org) and SNMP and associated MIBs.

## 5.3 Infrastructure

We will need substantial infrastructure to support the use of configurable run-time architecture information in intrusion management. Willow provides much of the infrastructure, but it is tailored for intrusion response. It currently has no support for intrusion detection or computer forensics, for example. For the infrastructure, we recognize a number of important problems that need to be solved.

- Not all application software is designed to be reconfigured. We are addressing this problem in Willow through a combination of standardized APIs and explorations of specific architectural styles (such as J2EE) for which reconfiguration is possible [17].
- Maintaining fidelity between running systems and the models is difficult because of the potential speed with which the running system's state can change.
- The ability of design-time architecture models to represent run-time information is still somewhat speculative. Prototypes exist, but have not been widely applied.
- It would be desirable to have a common modeling notation for all the models we have identified. Existing models use a wide variety of notations. We are examining RDF [12] and DAML [22] for this purpose.

## 6 Related Work

The Intrusion Tolerant Architecture project [20] at SRI is one project that is making explicit use of architecture information for security purposes. However, their goal is to analyze specific architecture styles at design time to verify selected properties relating to intrusion tolerance. They do not appear to be making use of the architecture models at run-time and they do not appear to be addressing run-time reconfiguration.

Another SRI project, Emerald [2], uses an application specific monitor to extract application level information. While not apparently based on architecture information, it should be possible to modify Emerald monitors to use such information. It is even possible that some form of automated construction of such monitors could be achieved.

Many architecture description languages (ADLs) have been developed to model architectures. Examples of ADLs include C2SADEL [15], Darwin [14], Rapide [13], UniCon [18], Wright [1], and ACME [7]. An important new entry into this field is the University of California, Irvine (UCI), xArch ADL [6], which we are using in the Willow project. xArch is an extensible, XML-based, standard representation for describing architectural models and for precisely capturing the structure of a software system. xArch is unique in providing a simple base specification with an incremental set of extensions. With the exception of xArch (via Willow) none of these ADLs have as yet been applied to intrusion management.

CFEngine [3] is a system administration tool for keeping systems running using a homeostasis approach. It unfortunately embeds any architecture information in agents (i.e., scripts), and so it is difficult to extract them in the form of declarative models.

The SHIM intrusion detection system [11] is one of the first specification-based IDS. It compares a specification of the behavior of an application program to the specification. Behavior is defined by the trace of system calls or log entries emanating from the program. SHIM focuses on these traces and this limits its ability to detect intrusions. It is, however, targeting the correct level (application software), and a combination of its behavior specification with an architecture model would be a promising research target.

## 7 Next Steps

Willow represents the first step in applying configurable run-time architecture information to intrusion management. The next step is to pick an additional phase of the intrusion life cycle and explore the application of architecture information to that phase. Currently, we are looking at architecture-driven computer forensics as the most promising target. This is because the current tools for forensics appear to be quite low-level and so there is significant potential for improvement. Farther out, we can explore adding architecture to an intrusion detection system. We have identified the UC Davis SHIM project [11] as a good starting point because it is a specification-based IDS targeting anomaly detection against application software systems.

## 8 Acknowledgements

This material is based in part upon work sponsored by DARPA, AFRL, ARO, and SPAWAR under Contract Numbers F30602-00-2-0608, F30602-01-1-0503, F49620-01-1-0282, DAAD19-01-1-0484, and N66001-00-8945. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

## 9 References

[1] Allen, R. and D.Garlan. "A Formal Basis for Architectural Connection". *ACM Transactions on Software Engineering and Methodology* 6(3): 213-249 (July 1997).

[2] Almgren, M. and U. Lindqvist. "Application-Integrated Data Collection for Security Monitoring". In *Recent Advances in Intrusion Detection (RAID 2001)"*. pp. 22-36. Davis, California, Oct. 2001. MONTH = {October}. Springer (pub.) LNCS 2001.

[3] Burgess, M. "Computer Immunology". In *Proc. of the 12th Usenix Systems Administration Conf.* p. 283. 1998.

[4] Coroner's Toolkit Web Page. http://www.porcupine.org/forensics/tct.html.

[5] Curry, D. and H. Debar. "Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition". IETF Internet Draft. Dec. 2001. *http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-06.txt*

[6] Dashofy, E., A. van der Hoek, and R.N. Taylor. "A Highly-Extensible, XML-Based Architecture Description Language". Proc. of The Working IEEE/IFIP Conference on Software Architecture, Amsterdam, The Netherlands, August 2001.

[7] Garlan, D., R. Monroe, and D. Wile. "ACME: An Architecture Description Interchange Language". In *Proc. of CASCON '97*. IBM Center for Advanced Studies. pp. 169-183. Nov. 1997.

[8] Hall, R., D. Heimbigner, and A.L. Wolf. "A Cooperative Approach to Support Software Deployment Using the Software Dock". In *Proceedings 1999 International Conference on Software Engineering*. Los Angeles, California, May 1999, pp. 174-183.

[9] Heimbigner, D., R.S. Hall, and A.L. Wolf. "A Framework for Analyzing Configurations of Deployable Software Systems", In *Proc. of the 5th IEEE Int'l Conf. on Engineering of Complex Computer Systems*. pp. 32-42. Las Vegas, NV, October 1999.

[10] Knight, J., D. Heimbigner, A. Wolf, A. Carzaniga, J. Hill, and P. Devanbu. "The Willow Survivability Architecture". Proc. of the Fourth Information Survivability Workshop (ISW-2001), 18-20 March 2002, Vancouver, B.C.

[11] Ko, C., P. Brutch, J. Rowe, G. Tsafnat, K. Levitt. "System Health and Intrusion Monitoring Using a Hierarchy of Constraints". RAID 2001, pp. 190-203.

[12] Lassila. O. and R.R. Swick (ed). "Resource Description Framework (RDF) Model and Syntax Specification". W3C Recommendation 22 February 1999. *http://www.w3.org/TR/1999/REC-rdf-syntax-19990222*.

[13] Luckham, D. C., J.J. Kenney, L.M. Augustin, J. Vera, D. Bryan, and W. Mann. "Specification and Analysis of System Architecture Using Rapide". *IEEE Transactions on Software Engineering* 21(4):336-355 (April 1995).

[14] Magee J. and J. Kramer. "Dynamic Structure in Software Architectures". In *Proc. of the 4th ACM SIGSOFT Symposium on the Foundations of Software Engineering*. pp. 3-14. Oct. 1996.

[15] Medvidovic, N., D.S. Rosenblum and R.N. Taylor. "A Language and Environment for Architecture-Based Software Development and Evolution". In *Proc. of the 1999 Int'l Conf. on Software Engineering*. pp. 43-54. May 1999.

[16] Perry, D.E. and A.L. Wolf. "Foundations for the Study of Software Architecture". SIGSOFT Software Engineering Notes, pp. 40-52. Oct.1992.

[17] Rutherford, M., K. Anderson, A. Carzaniga, D. Heimbigner, and A. L. Wolf . "Reconfiguration in the Enterprise JavaBean Component Model". Department of Computer Science, University of Colorado, Technical Report CU-CS-925-01, December, 2001.

[18] Shaw, M., R. DeLine, D.V. Klein, T.L. Ross, D.M. Young, and G. Zelesnik. "Abstractions for Software Architecture and Tools to Support Them". *IEEE Transactions on Software Engineering* 21(4): 314-335 (April 1995).

[19] Staniford-Chen, S., B. Tung, and D. Schnackenberg. "The Common Intrusion Detection Framework (CIDF)". In *Proc. of the 1st Information Survivability Workshop*. Orlando FL, October 1998.

[20] Stavridou, V. Intrusion Tolerant Architectures Project Web Page. SRI International. *http://www.sdl.sri.com/projects/itarch*.

[21] Tripwire Corp. Web Page. *http://www.tripwire.com/*.

[22] van Harmelen, F., P.F. Patel-Schneider, and I. Horrocks (ed) . "Reference description of the DAML+OIL (March 2001) ontology markup language". *http://www.daml.org/2001/03/reference*.

# Appendix A. Architecture and Environment Sub-Models

In order to support proper reconfiguration at run-time, it is necessary to identify the full range of sub-models comprising the architecture. We have begun this process as part of the Willow project, and currently recognize the following sub-models,.

**Family Model**: The family model represents the range of legal configurations of a software system architecture. It specified the possible configurations over both the artifacts that comprise a software system as well as the run-time components (clients and servers, for example).

**Deployment Model**: The deployment model represents a specific instance out of the configurations defined by the Family model. The choice is controlled by external properties from the environment into which the system is deployed. This particular model is one that is often overlooked because it represents the structure of the system at the level of deployed artifacts and thus is not always recognized as having a model. For an example of this, refer to the University of Colorado Software Dock project [8].

**Activation Model:** The activation model represents the running system. This model is a derivation of both the Family model, which specifies the components, and the deployment model, which specifies the actual files containing the code for those components. This model reflects the actual executing components and their connections over time. This model is also constructed through reference to it operational environment. This model also has significant dependency information associated with it to define, for example, startup/shutdown dependencies.

Although we will not detail them here, we also have identified several environmental models whose definition is needed to accompany our architecture models. These include a Host model, a Network model, and an Administrative model.