

Negotiating Posture Boundaries

Judith A. Stafford

Software Engineering Institute

Carnegie Mellon University

Pittsburgh, PA 15213 USA

jas@sei.cmu.edu

August 15, 2000

1 Introduction

Survivability of systems used for critical purposes is of increasing concern as the types of software used in these applications are developed to be used in unbounded, or widely bounded, networks. When software systems exist and function on an unbounded network they may access information and functionality provided by sources with various degrees of familiarity. The properties of such a system, therefore, may be dependent upon the properties of services that are provided by unreliable sources or accessed over insecure and/or unreliable channels. In this paper we propose a method for automated system posturing in the face of suspected intrusion or other form of system compromise. Conceptually, a tool based on this method would respond to a threat by creating a temporary fence, called a posture boundary, around a subset of a system's components. Temporarily bounding the system in this way allows it to continue operation with the maximum functionality possible given the type of attack that has occurred.

2 System Posturing

Effective and efficient hardening of an unbounded system requires the establishment of posture boundaries. A posture boundary is a communication boundary within a trusted environment. A posture boundary is established with respect to a given capability and surrounds the entire set of system components required to guarantee continued service in the face of a specified class of attack. Determining the set of components that must be contained

inside the posture boundary is a hard problem. All pieces of the system's infrastructure required for continuous operation at the agreed upon, reduced level of functionality must also reside inside the boundary. Transitive inter-component relationships must be identified with respect to the components directly required to support the given capability. It may be that these transitive relationships reach far from the core of required components and may require the inclusion of more infrastructure than is desirable when under attack. When this is the case it is useful to be able to tradeoff between risk and isolation.

2.1 Risk vs. Isolation

Given a capability, the method determines the entire set of transitively-related components. After this set of components has been identified, the tool interacts with the security team to perform a tradeoff analysis. There are two types of decisions that need to be made in order to ensure that the desired infrastructure will be continuously available:

1. Are there services provided by any components that can be cached inside the posture boundary?
2. Is it possible to narrow the scope of the capability in order to remove some of the required components from the set?

Item 1 refers to services such as the backup copies of files that have been obtained from outside the boundary. Item 2 refers to the determination that the cost of providing continuous support outweighs the benefit for some aspect of the capability. Such a decision might result from recognition that a marginally required aspect of the capability requires inclusion of a component that requires information from an unfamiliar website in which case it would best be left outside the posture boundary and the related capability be rendered temporarily unavailable. A tool based on our technique supports making such tradeoff decisions.

2.2 Identifying Posture Boundaries

Identifying posture boundaries requires knowledge of system components, the resources they use, the services they provide, and the connections among them. We have developed technologies that, when combined with deployment technology, can provide the infrastructure for identifying and automatically enforcing posture boundaries. A description of deployment view of the system's architecture is provided in the form of a deployable system description (DSD) [2] is used as a base for analysis. Components are annotated with security related properties [1] and intra-component pathways [4]; then architecture-level dependence

analysis [3] is used to identify transitive dependencies among system components with respect to capabilities representing the negotiated minimum desired level of functionality.

3 Conclusions

In this position paper we have suggested a technique for improving continuous access to critical functionalities of a system that has been identified as being compromised in some way. The suggested solution draws on technologies that have been developed with other applications in mind; however, we feel that they supply the right mix to support the development of an automated system posturing mechanism that can efficiently and rapidly assess the degree of system isolation required in order to support continuous access to system information and functionality.

REFERENCES

- [1] F. Bachman, L. Bass, C. Buhman, S. Cornella-Dorda, F. Long, J. Robert, R. Seacord, and K. Wallnau. Volume II: Technical Concepts of Component-Based Software Engineering. Technical Report CMU/SEI-2000-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, May 2000.
- [2] R.S. Hall, D. Heimbigner, and A.L. Wolf. A Cooperative Approach to Support Software Deployment Using the Software Dock. pages 174–183, Los Angeles, California, May 1999.
- [3] J.A. Stafford and A.L. Wolf. Architecture-Level Dependence Analysis in Support of Software Maintenance. In *Proceedings of the Third International Software Architecture Workshop*, pages 129–132, November 1998.
- [4] J.A. Stafford and A.L. Wolf. Annotating Components to Support Component-Based Static Analyses of Software Systems. In *Proceedings of Grace Hopper Conference 2000 (to appear)*, Hyannis, Massachusetts, September 2000.