Survivability of Computers and Networks based on Attack Signatures*

Axel Krings, Scott Harrison, John Dickinson
Computer Science Dept.
University of Idaho, Moscow ID, USA
{krings,harrison,johnd}@cs.uidaho.edu

Miles McQueen
INEEL
Idaho Falls, ID, USA
amm@inel.gov

Abstract

This paper describes a joint project between the University of Idaho and the Idaho National Engineering and Environmental Laboratories (INEEL) which addresses solutions to the problem of surviving malicious intrusions in networked computer systems. A new approach to survivability is proposed, that consists of a survivability architecture and run-time attack detection, both based on attack signatures. An off-line derivation process generates the attack signatures, which are then used in a bottom-up fashion as an aid to harden the system using methods and algorithms found in restrictive distributed fault-tolerant systems based on the principles of redundancy. Thus, the focus is shifted from intrusion avoidance and detection, to guaranteeing that critical functions survive even in the presence of intrusion. At run-time, malicious acts can be masked, while intrusion detection using attack signatures is used for attack recognition.

1 Introduction

Research dealing with malicious acts can be partitioned into three areas, namely Resistance, Recognition and Recovery [1]. Resistance encompasses traditional mechanisms that make it difficult for a hacker to infiltrate a system. Recognition on the other hand deals with intrusion detection. Resistance failed and intrusion detection methods attempt to observe the malicious act by detecting unusual system behavior. Traditionally, this is done with the help of log files, e.g. the log files are scanned to match known sequences of previous attacks. Lastly, recovery addresses the issue of surviving malicious acts. We define survivability as a combination of recognition and recovery steps. It should be noted that recognition can be omitted if fault masking is the only survivability objective. Many solutions found in fault tolerant system design are suitable for adaptation in order to increase survivability. Examples of such solutions are the introduction of time and information redundancy as well as space redundancy. We propose an approach that looks beyond recognition, i.e. intrusion detection, and extends the scope to computer and network survivability [6].

There are many commonalities between the area of fault-tolerance and network security/survivability. One main goal of fault-tolerant system design is to mask hardware faults, such that failure of components does not affect the ability of the system to perform to its specifications. Several fault models exist in which faults are classified by their behavior and relative probability of occurrence. For example, the likelihood of self-evident faults, i.e., benign faults, is much greater than that of malicious faults [7], i.e., Byzantine or asymmetric faults. The basic goals of fault-tolerance match very closely issues of system survivability and network security in distributed systems. However, whereas in fault-tolerant systems malicious faults are the least likely, in network security the attacker is expected to behave maliciously. Thus the probabilistic arguments and assumptions of fault-tolerance generally do not carry over to survivability and network security, e.g. the assumption that benign faults are the most likely [7] would incorrectly correspond to assuming that benign intrusions are the rule. However, the basic goal of fault-tolerance and survivability are essentially the same. As a result we will use the terms fault-tolerance and survivability interchangeably during the remainder of this paper.

Case studies of ultra reliable fault-tolerant and survivability systems have proven expensive in terms of overhead [3]. A more realistic approach to achieving survivability is to focus on the identification of critical functionalities, and not the complete systems. Such an approach was taken in [2], where a survivability map identifies the critical functionalities, together with existing and augmented survivability features in a top-down fashion. We submit a reverse approach, i.e. bottom-up, by focusing on critical functionalities as identified by means of attack signatures.

*This work has been supported by the Idaho National Engineering Laboratories (INEEL).

This proposal presents a survivability architecture as well as attack detection scheme. Section 2 gives the motivation for the approach taken. Next, Section 3 describes the concept of attack signatures. Section 4 concludes the paper with a summary and future research.

2 Motivation

In the recent Byzantine Resilient Approach to Network Security (BRANS) project [5], a top-down approach was used to achieve survivability. Critical functionalities were identified in a Linux system, e.g. DNS, telnet and passwd. These critical functionalities were then investigated at the code level and decentralized, i.e. redundancy was applied in order to mask possible alterations resulting from intrusions. The operating system was minimally modified with calls to, and returns from, an isolated module, the so-called Byzantine Agreement Module (BAM). The BAM was the central module on each computer that dealt with management and specifics of defined critical functionalities as well as all communication to BAMs on different systems participating in the replication. The concept can be seen in Figure 1, where the operating system is executing functionalities or modules m_i on processor p. In the unmodified operating system, m_i simply executes and terminates. However, here m_i is modified to call the BAM module. The BAM on processor p is interfaced with a service module which implements any code specific to the modification of m_i. Service modules m_i^p are in one-to-one correspondence with their critical functionalities, i.e., service module m_i^p maps to functionality m_i on processor p. As a result, the BAM on processor p may have a service module m_j^q which corresponds to functionality m_j on processor q. After receiving a call from m_i and under considerations of the specifics in m_i^p, the BAM initiates an agreement with all processors involved in the replication of the functionality. This set of processors is called the survivability cluster of m_i and need not be the same for all functionalities. BAM forwards the request of m_i to all processors in the survivability cluster which in turn execute their corresponding service modules. At this point in time each BAM in the cluster derived a result. Next, all BAMs in the survivability cluster initiate voting algorithms. The voted upon result is returned, processed, and a return from the BAM to m_i finishes up the execution.

Whereas the principle just described achieved the desired survivability of the functionalities, several practical difficulties were encountered during the specification and implementation:

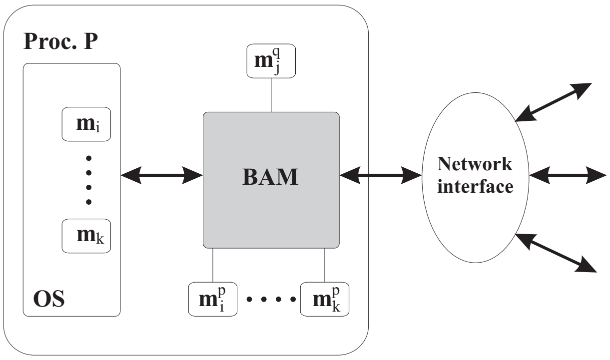

Figure 1: BRANS Concept

1. Understanding the operating system (OS) code is difficult, partially due to its complexity and size.
2. The ad-hoc approach addressing functionalities in their entire completeness was too complicated. It was very easy to overlook OS features. For example, changes in the environment constantly exposed new specifics, e.g. password versus shadowed password files, console versus remote access.
3. Frequent OS revisions, e.g. new Linux kernel releases, deemed the approach unmaintainable as the modifications were not automatic, but ad-hoc. Upgrading the OS assumed full understanding of the changes from one OS release to the other.
4. Critical functionalities are difficult to understand fully, i.e. identifying the exact source of the potential vulnerabilities in a top-down fashion was straightforward only in the simplest cases.
5. Agreement algorithms were often overkill. Most replicated functionalities only required simple majority rather than agreement.

Our conclusion to these problems is that in a changing environment a more flexible and automated approach to survivability is needed. Thus we investigate the suitability of a bottom-up approach to identify critical functionalities in a semi automated fashion.

3 Survivability using Attack Signatures

The survivability features of our approach are mainly achieved a priori and not at run-time. However, once the survivability mechanisms are in place, they will be used at run-time for attack analysis and possible adaptation. Thus, our approach has off-line and on-line components. The basic concept can be described as follows:

First, the operating system and/or applications are instrumented in order to profile the system. This allows recording the "signature" of a normal acting system. Profile based signatures, called usage models, are based on Markov models. Intuitively, usage patterns represent the frequency spectrum of the functionalities or functions. They have been used in intrusion detection, and it has been proven to be very difficult to quantify the "normal" use of the system in order to detect the "abnormal" use. It should be pointed out at one of the main difficulties of usage pattern analysis in intrusion detection is the determination of the threshold that differentiates normal from abnormal uses. Fixed thresholds do not consider the normal changes in behavior of software and users in a dynamic real world environment. The results are frequent false alerts, which undermine user confidence in the method. Dynamic or adaptive thresholds on the other hand may allow an attacker to slowly change the behavior of the system such that its usage patterns slowly drift towards the malicious goals. It is not our attempt to try yet another way of applying statistics to differentiate a normal from an abnormal usage patter, but to use signatures of attacks in order to identify the functionalities within the software that need to be made survivable a priori.

Next, an automated approach is used to identify the effected functionalities of the idle system while it is subjected to specific attack scenarios. The deviation of the normal system from the attack scenario is the attack signature of the attack. Readily available attack suites are decomposed into atomic attack scenarios. With the help of these atomic attack scenarios a library of different attack signatures is established with the help of customized attack simulations.

3.1 Attack Signatures

Whereas in practice attacks are mounted using attack suites, e.g. Satan, CyberPack, or LORNuker, we assume attacks to be atomic. An atomic attack A_i is the smallest attack technology unit, e.g. a port sweep. Thus an attack suite can be viewed as a collection of individual atomic attacks A_i. Limiting the scope of an attack to atomic units allows us to focus on a very narrow set of effected functions in the OS or application.

The attack signature corresponding to A_i will be denoted by $S_i(t)$. Signature $S_i(t)$ is a vector of length n, where n is the number of instrumented functions and t is the time duration of the measurement during the attack. Element j of $S_i(t)$ indicates the frequency $f(t)$ of function F_j contributed to by A_i. Thus $S_i(t) = (f_1(t), f_2(t), ..., f_n(t))$, and frequency $f_j(t)$ is directly related to the profile generated by function F_j. However, $f_i(t)$ is not simply the profile of F_j, but the normalized profile. As such, $f_j(t)$ does not reflect profiling information of an idle system.

The signature of the idle system is of special significance and will be denoted by $S_0(t)$. It constitutes the frame of reference and will be used to normalize attack signatures. Since the system is assumed to be idle, $S_0(t)$ is not affected by applications or user behavior. However, $S_0(t)$ is platform dependent, since in heterogeneous systems different hardware normally assumes different drivers supplied by the operating system.

If the raw profile generated by function F_i is denoted by $P_i(t)$, then the normalized attack signature $S_i(t)$ of an attack A_i is defined as $S_i(t) = P_i(t) - S_0(t)$. In the following we always assume signatures to be normalized. In order to capture the changes in the signatures we revert to the derivative of $S_i(t)$, i.e. $S_i'(t)$.

Given the fact that n is large, but the functions involved by specific atomic actions is relatively small, $S_i(t)$ is a sparse vector. Only non-zero elements are of interest.

A portion of a sample attack signature can be seen in Figure 2. The attack is a simple sequence of unsuccessful logins. Whereas the F_i axis indicates the modules profiled, the $f_i(t)$ axis gives the frequency resulting from profiling. Following the time axis, one can see how $S_i(t)$ changes over time. Around $t = 7$ one can observe a steep increase in $S_i(t)$. This "mountain range" is the impact of the sample attack and it peaks out at $t = 8$. The flat areas before and behind the range indicate that the system behaves close to the idle profile $S_0(t)$. The slice of Figure 2 at time $t = 9$ is the attack signature $S_i(t)$ of the attack and it is extracted in Figure 3.

3.2 Survivability Handlers

The attack signatures $S_i(t)$ and $S_i'(t)$ are used to identify those functions that are affected by the attack A_i. Inspection of the functions F_j, specifically the argument list, attempts to identify data and data structures of specific interest, e.g. password related information, user and group id (uid/gid) or strings. Only at this time is it necessary to modify the behavior of critical functions by applying survivability methods in order to increase the probability of avoiding malicious behavior, or recovering from it. In order to implement recovery, survivability handlers are injected in the code that may implement techniques like codes, authentication, fault-tolerant agreement, roll-back or roll-forward strategies. For example, if the data item is suitable for decentralization, i.e. if space redundancy is applicable,

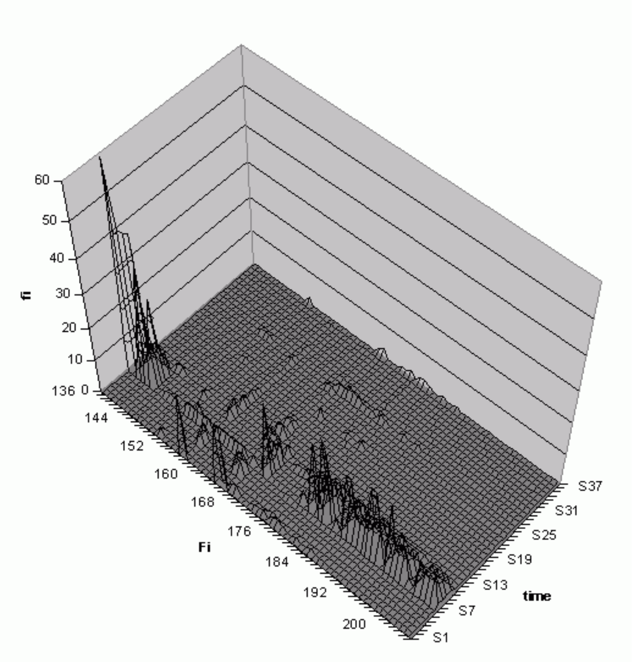

Figure 2: Partial Signatures in Time

agreement algorithms can use a communication framework similar to the BAM module of Figure 1 in order to achieve fault masking. In our current implementation the overhead of Early Stopping Agreement [4] among up to 7 hosts can be achieved within 1 to 100 milliseconds, depending on the system load.

4 Summary

This paper outlines an approach designed to identify functionalities affected by malicious attacks. Attack signatures are derived that help isolate operating system or application functions. These functions are then investigated for suitability of applying space, time, or information redundancy.

In the next step of the project survivability handlers are injected into the code. Based on the principles of redundancy, fault/attack masking are the goal.

Acknowledgment The authors want to thank Kosuke Imamura, Robert Morris, Gang Xiao and Shankar Kelath for their efforts and support in writing the necessary software tools.

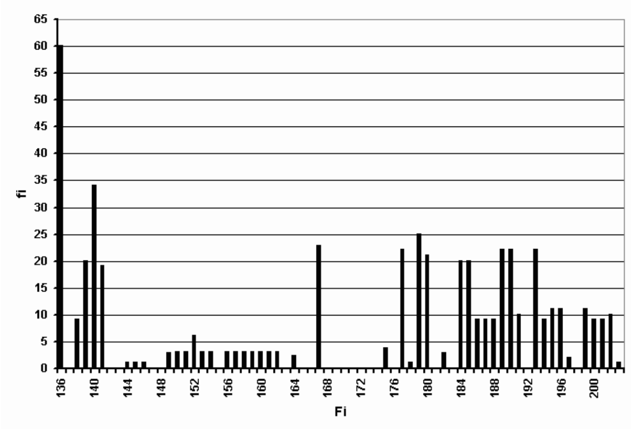

Figure 3: Partial Attack Signature S_i

References

[1] Ellison, et.al., "Survivable Network Systems: An Emerging Discipline", CMU, Software Engineering Institute, *Tech Report CMU/SEI-97-TR-013*, revised May 1999.

[2] Ellison, et.al., "A Case Study in Survivable Network System Analysis", CMU, Software Engineering Institute, *Tech Report CMU/SEI-98-TR-014 ESC-TR-98-014*, September 1998.

[3] Kieckhafer, R.M., et al, "The MAFT Architecture for Distributed Fault-Tolerance", *IEEE Transactions on Computers*, V. C-37, No. 4, pp. 398-405, April, 1988.

[4] Krings, A.W. and T. Feyer, "The Byzantine Agreement Problem: Optimal Early Stopping", *Proc. HICSS32, Dependable Distributed Systems*, paper *stdds03*, Jan., 1999.

[5] Krings A.W, and M.A. McQueen, "A Byzantine Resilient Approach to Network Security," *Proc. 29th International Symposium on Fault-Tolerant Computing, Fast Abstracts: (FTCS-29)*, Madison, Wisconsin, USA, June 15-18, 1999, pp. 13-14.

[6] Peter G. Neumann, "Practical Architectures for Survivable Systems and Networks: Phase-One Final Report", Computer Science Laboratory, SRI International, January 1999.

[7] Thambidurai, P., and You-Keun Park, "Interactive Consistency with Multiple Failure Modes", *7th Reliable Distributed Systems Symposium*, Columbus, OH, pp. 93-100, October 1988.