On Intrusion Resiliency

Tim Lawless, CISSP <lawless@wwjh.net>

Abstract

This paper puts forth the concept of intrusion resiliency as an emergent behavior that occurs within coupled intrusion detection and intrusion response mechanisms when the mechanisms, as a whole, exhibit a key set of identified attributes. In illustrative example of how these attributes interact with each other to produce this behavior is given in the form of the Saint Jude Linux Kernel Module.

Introduction

During recent years, significant strides have been made in the identification and elimination of software flaws that open up windows of exposure during which the integrity of host systems may be assailed. Unfortunately, the increased awareness and attention that security-related software flaws have drawn has come with a cost. The rate at which new vulnerabilities are being discovered and published is increasing at super-linear rates, according to data compiled by CERT/CC[i], and SecurityFocus.com[ii] (Figure 1). This growth directly translates into more frequent windows of exposure during which host systems are susceptible to being compromised.

A Window of exposure opens for host systems of a particular class each time vulnerability is discovered[1]. In the case of unpublished or private vulnerabilities, these windows of exposure don't readily close, and may be exploited extensively prior to the knowledge of their existence becomes public[2]. The "threat-space" that these unpublished vulnerabilities occupy will likely expands in proportion to the space occupied by public vulnerabilities as a result of new movements within the underground to conceal vulnerabilities from public disclosure[iii iv].

[1] This does beg the eternal question: If a buffer overflow occurs in a trusted binary and no researcher is around to discover it, is there vulnerability?

[2] And still, after it is known that a vulnerability exists – the task remains to identify where the vulnerability is, and subsequently identify a resolution to the vulnerability while not effecting mission critical services.

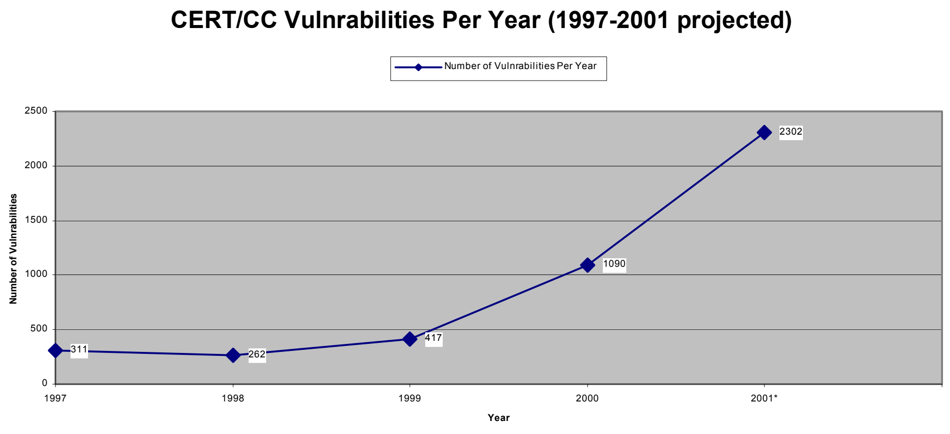


Figure 1 – Number of Vulnerabilities/Year

When faced with these new, and likely unpublished, vulnerabilities -- and their associated exploits -- the various perimeter-based defensive mechanisms may offer little or no protection. Without known vulnerability information to use in populating signature databases, analysts protecting hosts systems are forced to wade through alarms generated by anomaly detectors -- weeding out the new and unknown attacks from the background noise.

Advancing the threat further are new classes of improved delivery mechanisms such as the worm strategy first developed at Xerox Parc[v], later employed by R. Morris[vi] and reborn in recent months in the form of both UNIX and Windows worms[vii] [viii]. Advancements and new theories within the attack-delivery community promise to reduce the time needed to expend the population of vulnerable hosts from days to hours[ix].

To address these issues, the concept of survivable systems is emerging in the hopes of producing classes of hosts that can survive an attack against present but unknown vulnerabilities that may be delivered via an unknown but aggressive attack vector, all the while maintaining an acceptable level of operation[3]. Intrusion Resiliency, as a concept is a subset of the survivable systems (WORD), that attempts to identify intrusion incidents prior to their occurrence, but at extremely short distance from the incident occurrence.

Timeline of an Intrusion Incident

Typically, when an intrusion attempt occurs, an assailant targets a system with an end-game goal for the intrusion attempt. The assailant wishes to acquire either an access privilege or piece of information not previously held. Ultimately, the goal is achieved by performing a series of events over a period of time that allows the assailant to circumvent

[3] And that is a mouthful.

any measures that may normally impede their activity[x]. The final event that transitions a host system into a compromised state, signaling the achievement of the assailant's goal, is the intrusion event. One may conceptualize the sequence of events that lead up to the intrusion event, and the events that follow may a timeline of an intrusion incident.

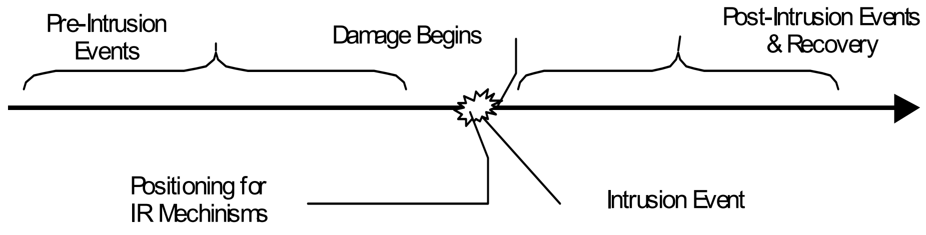

Pre-Intrusion Events
Damage Begins
Post-Intrusion Events & Recovery
Positioning for IR Mechinisms
Intrusion Event

Figure 2 – Timeline of an Intrusion

The pre-intrusion event time period is spent conducting intelligence-gathering operations, staging and delivering of the attack that generates the intrusion event. Typically this is the period of time that intrusion detection mechanisms are desired to detect the pre-cursors to the intrusion event, such as port scanning, failed authentication attempts, or file access failures.

During the pre-intrusion time span, the universe of potential sequences of events that may lead to the actual intrusion event decreases as the assailant advances towards their goal. The sequences that emerge contain behavior that may be described as a Markov Chain, with the set of possible next-steps diminishing as the assailant progresses towards their goal.

On the other end of the timeline of an intrusion, the potential for damage begins upon the conclusion of the intrusion event and continues to accumulate until such time that a response can be mounted to contain and extenuate the incident. Methods to contain this damage through compartmentalization of process-bound resources exist; however within the compartment, the intrusion is unfettered, and any trust relationships or communication channels between compartments represent possible vectors by which the intrusion may further spread – compromising additional compartments.

After containment, recovery begins where the damage that was caused is assessed and repaired. It is important to note that an intrusion, in itself, does not cause damage; it is the means by which damage latter occurs via the loss of integrity, confidentially, or availability. If it were possible to detect and neutralize an intrusion before the potential for damage presented itself, then the impact of the intrusion would be the same as if no intrusion ever occurred.

Intrusion Resiliency

Quite simply, Intrusion Resiliency is the emergent behavior of a system that results from the introduction of a security mechanism that permits the host system to detect the presence of ongoing and successful attacks against vulnerabilities, known and unknown, and subsequently adjust the host's behavior in such a way as to neutralize the attack. By a similar token, the mechanisms that endow their host-system with this behavior are themselves an Intrusion Resiliency System.

Within the timeline of an intrusion, the Intrusion Resiliency System will assert itself during the actual intrusion event. In comparison, traditional protective mechanisms attempt to inhibit the attack, and detective mechanisms pick up on latent artifacts of the intrusion.

Intrusion Resiliency systems are comprised of a detector and a responder. The detector may be based on currently existing or emerging intrusion detection methodologies, but must operate with a level of certainty that will permit a response without outside intervention or oversight. The responder will, by any means, terminate, divert, or otherwise neutralize the detected intrusion activity.

The Intrusion Resiliency behavior of the detector–responder combination seemingly emerges when the resultant mechanism exhibits certain attributes critical to achieving the resiliency to an intrusion. These attributes are Independence, Immediacy, Intercedency and Dominance.

Independence

The mechanisms of the intrusion resiliency system must be capable of operating independent of external intervention once deployed. The sources of any external intervention represent a trusted source that would be able to affect the operational performance of the intrusion resiliency mechanisms, opening up a channel by which the mechanisms' protection could be neutralized. This requirement has several implications on the nature of the detective mechanism that may be used.

The isolated and automated nature of the mechanisms requires that the detection mechanism must exhibit an exceptionally low level of false positives. This precluded traditional anomaly based detectors that require external verification of their results by an analyst.

Further, regular updates to a database of known attacks or patterns of misuse can not be presumed – hence, Misuse Signature Detection is not usable. Without the ability to update a rule-base of known attacks, the effectiveness of these detectors degrades as a function of time.

To achieve a high level of accuracy within the detection mechanism, while successfully detecting new or unknown attacks, and maintain an independence from external sources, new hybrid methodologies such as model based or state-transition based detection engines may prove to be appropriate[xi]. The current examples of these emergent detection methodologies yield a higher, and more acceptable, accuracy in their detection – though, at the additional cost of needing to perform extended sampling of behavior on the target system to be protected. In the case of some detection methods[xii], slight deviations within the target system's configuration can cause the target system to generate false positives until such time that a resampling is performed.

Immediacy

While events occur within the target systems that are monitored, the detection mechanisms must make immediate determination that the events are acceptable, or indicative of an intrusion event. In the event that the event is flagged as an intrusion event, then a response must be initiated. Immediacy may be implemented as either temporal immediacy or sequential immediacy.

Immediacy is ideally temporally immediate, but also may rather sequentially immediate. As long as any state changes that occur as a result of the intrusion event are contained to the system on which the event occurred, and that system is not able to effect or initiate a state change on other, external, systems – then sequential immediacy is acceptable. This sequential immediacy may be implemented through a mechanism, such as a state-rollback to the state just prior to the state identified with the intrusion event.

The author has only experimented at length with temporal immediacy; however work has been done in an attempt to achieve sequential immediacy. A brief analysis was undertaken to determine the ease or benefit of using sequential immediacy; however, initial results appeared to indicate that true sequential immediacy would not be possible from an autonomous system – since the system would not truly be able to roll back its own state completely while maintaining a form of situational awareness to grant the system the ability to stave off the intrusion event upon the occurrence after the rollback.

Intercedency

The placement of the mechanisms that comprise the intrusion resiliency system is critical to achieving immediate and effective detection and response. By placing the mechanisms within a target system in such a way that the mechanisms have an opportunity to intercede during the intrusion event -- intercepting the intrusion before the attack objective is realized -- the host system may avert the damage associated with the intrusion. In strategy, this intercedency is similar to the NIDS practice of an application firewall where transactions are authenticated and validated before being passed on to vulnerable systems that the firewall attempts to protect vulnerable internal systems.

The placement of the mechanisms in temporal relation to the intrusion event may vary; the only requirement is that all paths along the attack-tree, which lead to the intrusion event, must be intercept-able. However, it may be found to be beneficial to temporally locate the mechanisms as near to the actual intrusion event's conclusion as possible, in order to minimize the uncertainty for which the detection mechanism must compensate, and subsequently the potential for erroneous results (false positive or negative).

Referring back to the attack tree methodology, as an attacker nears the goal; fewer remaining possible forward paths will exist towards the goal. Prior to the realization of the goal, all possible paths converge into a single path that leads to the intrusion event itself. Any appearance of atomism of an event is only an illusion that results from the level of granularity inherent in view of the attack model.

Dominance

When a method or device successfully inhibits attacks against a desirable target, it should be expected that the defensive mechanism or device would itself become a target of attacks. To be able to operate continuously in the face of such attacks, intrusion resiliency mechanisms must maintain dominance over any possible vectors of attack.

This attribute is the only of the three attributes that is transient. It should be expected that any single method of maintaining dominance over an attack on the Intrusion Resiliency mechanism would be overcome after an unknown period of time. To counter this problem, multiple defensive strategies may be deployed to improve the survivability of the intrusion resilience mechanism. Detailed below are but a few:

Secrecy[4]

This is the most tenuous, but also one of the most effective methods to achieve and maintains dominance of the intrusion resilience mechanisms. If an attacker is unaware of the nature and presence of the mechanisms, the task of countering the mechanisms is multi-fold times more difficult.

Once an attacker becomes aware of the presence of an Intrusion Resiliency mechanism, either by analysis of a failed attack or through out of band channels, the effectiveness of this strategy is nullified.

If this strategy is used, it is beneficial to minimize the amount of debugging information supplied to an attacker when an attack fails by concealing the mechanisms presence, introducing conflicting and erroneous data, or presenting the attacker with a simulated environment, permitting the intruder to operate under the false assumption that the attack succeeded.

Partnering

[4] It should be noted, for completeness, that this is not truly the use of 'obscurity' decried by many as giving a false sense of security. Secrecy, as a tactical tool, is only a force multiplier. If the strength of a security mechanism is nil, then the effect of secrecy in connection with that mechanism is equally nil. However, if secrecy and misinformation are used to augment the effectiveness of a security mechanism that provides a positive protection, then the multiplicative effect is truly useful in inhibiting an attacker from circumventing the protective devices while evading detection.

In some scenarios, it is not possible for a defensive mechanism to protect itself from subversion while at the same time protecting a target resource. In these situations, a level of protection may be achieved by bundling protective mechanisms together, where a portion of the mechanisms consider their counterpart mechanisms to be their target resource.

Each individual mechanism should be capable of operating autonomously of its ‘partner’, reducing the risk of a cascade effect if one resource should successfully be neutralized. If the partner’s purpose is not to protect, but to act as an agent for the recovery of its target resource, this tactic may prove even more profitable.

Out of Vector Placement

By placing the mechanisms outside the operational reach of attackers, such as behind the resource or system that is being protected, the mechanisms operate from a position of natural dominance.

Care should be taken to identify vectors where dominance by out-of-vector placement may be inverted. An example would be a kernel-based IDS mechanism being defeated by hostile code loaded via the kernel modules or DDI[5] interface on a Linux or Sun box. In cases where dominance inversion may occur, it is necessary to mitigate the risk through the employment of multiple dominance strategies.

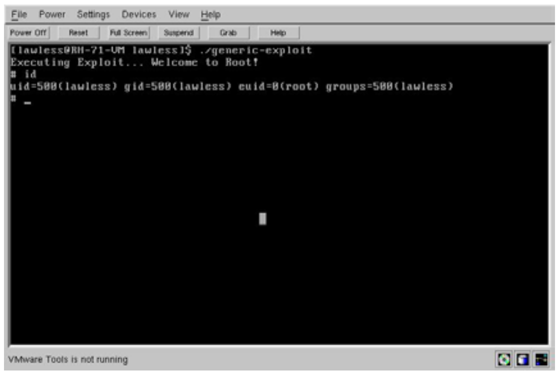
Saint Jude, Linux Kernel Module (LKM)

What follows is an illustration each of the preceding attributes of an Intrusion Resiliency System as they exhibit their selves in the Saint Jude Linux Kernel Module.

Background

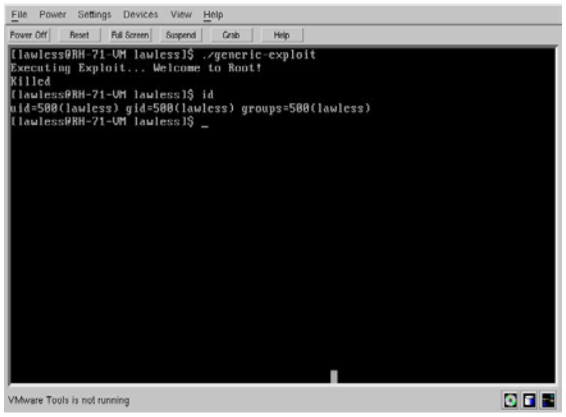
The Saint Jude LKM is a kernel-based intrusion detection system that achieves a level of intrusion resiliency for root-privilege escalation intrusions on Linux systems by monitoring privilege transitions. When running in a production environment, the Saint Jude LKM identifies root account privilege escalation attacks of known and unknown type, from sources internal and external to the host system, and neutralizes the attacks by aborting or diverting the attack. The detection is performed at the last possible moment prior to the protected system being transitioned into a root-compromised state by monitoring system activity from within the key system-calls that are member to a privilege-transition attack.

[5] Linux Kernel Modules and Sun Device Drivers execute a segment of code upon loading in order to initialize internal data structures. This code operates with the privilege of the kernel, and is not restricted in what actions it may actually perform. One possible action could be to utilize the elevated kernel privilege to disable, evade, or pervert intrusion detection mechanisms from operating. For more information reference, “kernel rootkits”


File Power Settings Devices View Help
Power Off Reset Full Screen Suspend Grab Help
VMware Tools is not running

```
[lawless@RH-71-VM lawless]$ ./generic-exploit
Executing Exploit... Welcome to Root!
# id
uid=500(lawless) gid=500(lawless) euid=0(root) groups=500(lawless)
# _
```

Figure 3 - Exploit Without St Jude


File Power Settings Devices View Help
Power Off Reset Full Screen Suspend Grab Help
VMware Tools is not running

```
[lawless@RH-71-VM lawless]$ ./generic-exploit
Executing Exploit... Welcome to Root!
Killed
[lawless@RH-71-VM lawless]$ id
uid=500(lawless) gid=500(lawless) groups=500(lawless)
[lawless@RH-71-VM lawless]$ _
```

Figure 4 - Exploit with St Jude

Independence

At the core of the Saint Jude LKM Intrusion Detection System is the Saint Jude Model. The Saint Jude model was designed to describe privilege transitions[6] within a UNIX environment and detect when a privilege transition occurs as the result of an intrusion event. A full definition of the model is available in "Saint Jude, The Model"[xiii]

The model operates by associating a set of allowable transitions to each privileged process operating within the system. The process is restricted to only performing the defined transitions. If an unprivileged process acquires privilege through one of the defined means, such as a setuid binary, a transition set is associated with based on the

[6] A "Privilege Transition" is a transition into or within a privileged state, account, or role. In the case of UNIX, the root account is an example of a privileged account. Processes within the running system perform transitions by 1) altering privilege state (acquiring, changing, or dropping), or 2) execution of a new application (and thereby transferring the privilege to the new application associated with the process).

means by which the process came into its privilege. The set of allowable transitions is only permitted to shrink, as to prevent circuitous attacks on the system's integrity.

The result of this 'rule based anomaly detection'[xiv] is that the rule base need only be updated when the system's configuration changes significantly. Any deviations from the rule base are treated as indicative of an intrusion, and require no external verification. The granularity of the model is such that slight modifications to the operational systems are possible without invalidating the model of the systems behavior. Such modifications include minor upgrades and patches to applications that require privileged access, and addition of new applications that do not require privileged access in order to operate.

Immediacy

The first implementations of the Saint Jude model operated on Solaris systems as a perl script that would monitor output from the BSM[xv] subsystem to identify intrusion events. In the field, this proved successful in detecting intrusions, but not in completely eliminating damage. The problems with the user-space perl implementation were not with the perl program itself; it was the way in which the perl implementation received its data and the associated latency.

The information was present; the problem was that the information was not immediately available. This window of opportunity that was opened for hostile agents, though short, was unacceptable. During those few seconds, a hostile agent could damage or destroy the running system, neutralize the intrusion detection mechanism, or insert methods of re-entry that would go undetected by the intrusion detection mechanism.

In analyzing the problems, the only apparent solution was to go into the kernel and get the data, instead of waiting for the data to emerge from the kernel via the BSM subsystem. Further, it seemed necessary that the actual analysis of the data would have to also occur within the kernel, since the latency in re-exporting the data to an external process would neutralize most of the benefits of having the data available immediately.

Interposition

The re-implementation of the Saint Jude model was done within the Linux kernel. The Linux system was chosen as the implementation candidate due to its open source code tree and ready information about the inner-workings of the kernel. Key system functions, which were identified in earlier implementations, were mapped to system calls within the Linux kernel. References within the kernel to the individual system calls were replaced with references to new wrapper functions. Within the wrappers functions, processing would be performed and the original system calls called.

One of the wrapper functions, the execve call, contained the actual analysis and response engines. This is the terminal system call before an intrusion occurs: with an intruder spawning an application (i.e., a command shell) on the remote host with the heightened privileges of the attacked application.

Interposing the analytical engine and the data collection points within the operating path of the attack provides the opportunity to determine immediately that an event is an intrusion, and respond before the intrusion event completes. If an intrusion is detected, the system call diverts execution away from the original system call, rather directing the execution to a response function that would terminate the offending process (and all of its associated processes) or redirect the execution to a program to do data collection for later forensic analysis.

Dominance

Already mentioned was the concern that the user-space perl implementation of the Saint Jude model experienced latency in receiving system status information. This latency directly translates into a delay in detection and subsequence response to any intrusion event. Once transitioned into the privileged state, an intruder would only need to transmit a SIG_KILL signal to the intrusion detection engine to terminate it. With the entire model implemented as a kernel module, terminating the detection engine becomes less trivial

The Saint Jude LKM operates wholly from within the kernel, and with the privilege and authority associated with ring zero software. From this position it is not normally vulnerable to attacks originating from within the protected systems' user space. By the design of the Linux operating system non-privileged processes are unable to affect the kernel, except through well-defined and validated points such as system calls. In some configurations it is possible for the root account to modify the kernel memory through the kmem device or by using other hostile kernel modules.

From the point of a non-privileged process, the Saint Jude module is out of bounds and inaccessible. The user processes are incapable of arbitrarily aborting, modifying, terminating, or otherwise neutralizing the module's activities.

Although the arguments passed to monitored system calls pass through the Saint Jude system call wrappers, application of strict bounds checking and scrubbing of data prior to any usage eliminates the possibility of an attack of the module via interface flaws. Further, mitigation of any unforeseen architectural issues may be managed so that errant calls to uninitilized memory result in a kernel panic, rather then transference of control to injected code.

For processes that operate with the root privilege, there are two primarily profitable vectors that may be used to attack an intrusion detection system operating within the kernel: direct modification of kernel memory or replacement of the on-disk copies of the kernel and modules. In the more recent work on Saint Jude's kernel module implementation, care has been taken to address each of these threats by (a) partnering with other mechanisms[7] to protect the kernel and intrusion detection engine by monitor the integrity of the kernel memory, (b) limiting the ability of root processes to directly modify kernel memory through the kmem device, (c) disabling the ability to modify system files critical to system boot, and (d) obfuscating detectable patterns in the engine's code that may be used to acquire a point of reference within kernel memory.

As a final means to strengthen the defensive posture of the module and the protected system, the module may optionally be configured to emit no reference to its activities within the system. In this mode, the module simply protects the system without generating output or status information. A redirected response would be necessary if a silent alarm is desired to be sounded, such as generating a SNMP trap or sending a message to a remote system.

Future work on concealment and misdirection will focus on run-time encoding and decoding of the on-disk and in-memory copies of Saint Jude, as to further blind potential assailments of the system.

Conclusion

With the rate of vulnerability development unlikely to show any signals of reversing the current acceleration, even if faced by proposed governmental regulations, the role of defending systems from hostile entities will be required to transform from its present state of detecting events after their occurrence, to a more proactive defensive posture embodied in the concept of Intrusion Resiliency. Saint Jude is but only a simple example of how a first-generation Intrusion Resiliency system may look and operate. As Time progresses, and with the addition of advancing artificial intelligence technology, Intrusion Resiliency systems will emerge that are simpler to configure, and operate over a broader definition of an intrusion.

[7] The Saint Michael Kernel Module was integrated into Saint Jude Kernel Module for the purpose of monitoring and defending kernel integrity after activation. Named after the Archangel Michael, defender of heaven and patron saint of guards and law enforcement officers, the Saint Michael Kernel Module monitors various portions of the kernel's text and data sections for indications of modifications that are caused by rootkits attempting to perform acts of concealment.
The Saint Michael Kernel Module may be found at http://sourceforge.net/projects/stjude

[i] http://www.cert.org/statistics
[ii] http://www.securityfocus.com/vulns/stats.shtml
[iii] http://anti.security.is
[iv]http://www.microsoft.com/technet/treeview/default.asp?url=/technet/columns/security/noarch.asp
[v] *The "Worm" Programs – Early Experience with a Distributed Computation*. John F. Shoch and Jon A Hub, Xerox Palo Alto Research Center.
[vi] http://www.worm.net/
[vii] http://www.whitehats.com/library/worms/lion/
[viii] http://www.eeye.com/html/Research/Advisories/AL20010717.html
[ix] "Warhol Worms, The Potential for Very Fast Internet Plagues
http://www.cs.berkeley.edu/~nweaver/warhol.html
[x] Counterpane, Attack Trees (Doctor Dobbs Journal, December 1999)
http://www.counterpane.com/attacktrees-ddj-ft.html
[xi] T.D. Garvey and T. Lunt. *Model-based intrusion detection*. In Proc. 14th Nat. Computer Security Conference, pages 372--385, Washington, DC, October 1991
[xii] Anil Somayaji and Stephanie Forrest. Automated Response using System-Call Delays. In Proceedings of *the 9th Annual USENIX Security Symposium.*
[xiii] http://www.wwjh.net/stjude/StJudeModel-Rev-1.02.pdf
[xiv] Rebecca Bace, Intrusion Detection. Macmillan Technical Publishing, 2000.
[xv] http://docs.sun.com/ab2/coll.47.8/SHIELD/@Ab2TocView?Ab2Lang=C&Ab2Enc=iso-8859-1&DwebQuery=BSM&oqt=BSM