

## BATTLEFIELD INTRUSION DETECTION SYSTEM, VERSION 2<sup>\*</sup>

Cynthia D. McLain, David A. Kassay, Robert K. Cunningham  
MIT Lincoln Laboratory  
Lexington, MA

**Abstract**—*The US Army is deploying a system connected via a mobile, wireless network that collects, distributes and displays information about the surrounding battlefield. This information is valuable, so the computers that store and display it and the communication infrastructure that transmits it need to be protected from attacks. In this paper, we describe Battlefield Intrusion Detection System (BIDS), version 2, a host-based intrusion prevention system. BIDS v2 operates by mediating application to operating system interactions. It has a configurable reporting and reaction capability and has been integrated with FBCB2 v7.0.3.*

### I. INTRODUCTION

The Global Information Grid (GIG) is the foundation for the future vision of net-centric warfare. The GIG vision is widespread connectivity and an integrated information infrastructure in a networking environment that is dynamic, adaptive, robust, and secure. The 4<sup>th</sup> Infantry Division, the first digitized division in the US Army, used a mobile wireless network consisting of vehicles equipped with computers in the recent war with Iraq [1]. Those computers are connected to the Tactical Internet (TI), that provides access to tactical situational awareness information, and may also be connected to the Secret Internet Protocol Router Network (SIPRNET), which may contain strategic plans [2]. In a battlefield environment mobile vehicles maintain uncertain connectivity with the network and may be subject to overrun. Because of the sensitivity of the information, security systems must be able to ensure adequate protection in that environment.

The Battlefield Intrusion Detection System (BIDS) project strives to protect such mobile computer systems from active computer attacks. Active attacks manipulate the system and network to obtain data and control not normally shared with a regular user of a system. The other type of attacks, passive attacks, include monitoring an open connection or unlocked display to obtain information

for use against US and coalition forces. The impact of a passive attacker can be limited by requiring periodic user authentication and using network encryption. While modern commercially available operating systems provide some security against active attacks, they are still vulnerable to a knowledgeable computer attacker [3].

With BIDS v1 we demonstrated a system that detected and responded to active attacks by monitoring interactions with the operating system, detecting unauthorized privilege changes and common attacker actions, and reacting appropriately [4]. Although BIDS v1 accurately detects attacks, it is capable of supporting only damage control operations. We wanted to be able to go beyond damage control and actually prevent the attacks from succeeding.

In the following sections we describe BIDS v2, which both accurately detects attacks and prevents them from succeeding. BIDS v2 also has a response capability to implement more extensive prevention measures. Section II describes the BIDS v2 architecture, Section III discusses the implementation of that architecture, Section IV presents evaluation results, and Section V discusses related work.

### II. BIDS v2 ARCHITECTURE

BIDS is a misuse detection system which models attack behavior and bases attack detections on actions that match that underlying model. Misuse detection systems based on models are distinct from signature-based systems in that they can detect novel attacks. Anomaly detection systems such as [3] are not suitable because they are difficult to tune in an environment where sudden changes in activity patterns occur. Misuse detection systems can be tuned so they produce few false positives but are still able to accurately detect attacks including insider attacks. One limitation of misuse detection systems, such as BIDS, is that although they can detect novel attacks, they can detect only attacks for which a model exists.

The BIDS attack model is based on unauthorized usage of certain system calls. BIDS v1 accurately detected that type of misuse but was unable to prevent it. BIDS v2 prevents the attacker from misusing the system for the activities that BIDS is able to detect.

---

<sup>\*</sup> This work was sponsored by the US Army CECOM under contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Attacks that spawn command shells or interact with the file system require use of system calls implemented as functions in the operating system (OS) kernel. The BIDS v2 architecture uses system call mediation to intercept those attacks before they can succeed. In UNIX, access to the kernel functions from user space is supplied via a well-defined interface [5]. By adding a software shim on the kernel side of this interface, user space function calls into the kernel can be intercepted.

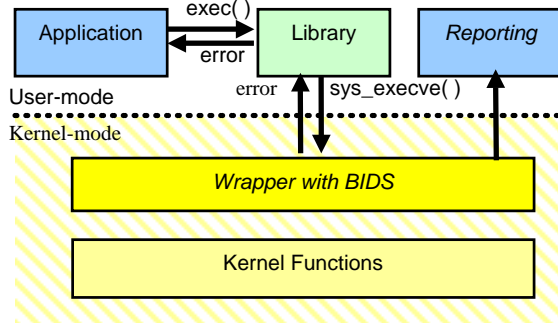


Figure 1. BIDS v2 architecture

As an example, in Figure 1, the software shim is the box labeled *Wrapper with BIDS*. In the figure, the application process issues an *exec* system call to begin execution of a new process. The user-mode system call is implemented in a system library that knows how to reference the kernel-mode version of the system call. An algorithm in the BIDS wrapper detects an attack in the arguments to the kernel-mode system call function, *sys\_execve*, and causes an error code to be returned to the application process and an intrusion alert to be issued by the reporting module. Processes are prevented from accessing kernel functions only when BIDS determines those functions would result in unauthorized activity.

BIDS v2 provides the configurable reactions and reporting found in BIDS v1. Also, BIDS v2 produces the same types of alerts as BIDS v1. Implementation differences between the two versions will be discussed below.

#### A. Generic Software Wrapper Toolkit

The system call mediation used in BIDS is implemented using the Generic Software Wrapper Toolkit (GSWTK) [6]. The GSWTK was developed by NAI Labs to explore event interception techniques that can be used on multiple operating systems. It provides several capabilities that are relevant to our project: system call mediation; an abstraction for writing the kernel functions used for wrapping system calls; and a communication mechanism between kernel space and user space.

The GSWTK required adjustments to work under newer versions of Linux and extensions to support BIDS v2 functionality. The short time-line we had for developing BIDS v2 required leveraging the work of others. The use of the GSWTK for BIDS v2 will be discussed in more detail in Section III. Alternatives to the GSWTK will be discussed in Section V.

#### B. Analysis and Detection

All but one of the BIDS algorithms, buffer overflow detection, makes use of the observation that all valid changes in ownership of a process require use of an authentication procedure. BIDS maintains information about the authenticated user of the process and compares that to information about the current user of the process to determine if the specified operation should be permitted.

In BIDS v1, the algorithms analyzed audit data produced by the Solaris Basic Security Module (BSM) [7]. Since the auditing system was implemented as part of the Solaris operating system, authentication information was provided in the audit records.

BIDS v2 is implemented on Linux, which provides no native auditing capability. Although third party auditing solutions are available for Linux [8][9], they don't provide the same capabilities as the BSM. Since establishing the authenticated identity of the user is critical to the functioning of the BIDS algorithms, BIDS v2 must obtain this information on its own.

The BIDS v2 architecture was also affected by migration of the BIDS intrusion analysis to the OS kernel. Since BIDS v2 can prevent a system call from completing, it is crucial to eliminate false positives without introducing false negatives. Also, since overhead is incurred with each wrapped system call, performance is critical. As a benefit, being implemented in the OS kernel provides BIDS with access to more information than is available via audit records, information which was used to improve some of the BIDS algorithms.

The same detection algorithms are used in BIDS v2 as in BIDS v1 but their implementations differ, as will be discussed later. The algorithms are summarized here.

**Bottleneck Verification (BV):** There are three algorithms in the BV group; all check for unauthorized creation of a command line shell. The alerts are distinguished by whether the unauthenticated user is a regular user or the super user or whether the access is via a group to which the authenticated user does not belong.

**Unauthorized Switch Owner Tool Creation:** The two algorithms in the switch owner group check for creation of files that allow the user to run the program with the privileges associated with the owner of the file. The alerts are distinguished by whether the privileges pertain to the user id or group id of the file owner.

**Persistent Object Monitoring (POM):** There is one algorithm in this group; it alerts whenever an unauthenticated user attempts to modify a protected file that would be inaccessible otherwise.

**Dynamic Object Monitoring (DOM):** The sole algorithm in this group alerts whenever an unauthenticated user tries to modify a protected file, one that would be inaccessible otherwise, via a symbolic link.

**Injected Code Detection:** At present there is one algorithm in this group. It alerts when an argument to a program execution instruction contains more than N non-ASCII characters. We are adding another algorithm to alert when a file containing malicious code is detected [10].

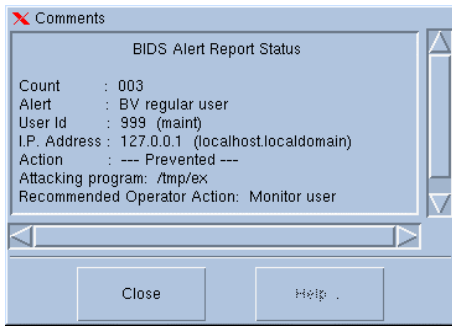


Figure 2. FBCB2 7.0.3 Security Manager dialog box showing BIDS alert

### C. Reaction and Reporting

While it's important to protect the system, it's also important to alert the security manager that the system is under attack. In addition to the reporting capabilities provided by BIDS v1, BIDS v2 provides an interface for reporting to the security management system on the FBCB2. Figure 2 shows an example of a dialog box with a BIDS alert, taken from the security management system developed by Northrop Grumman Mission Systems. The reaction capability was also retained in BIDS v2. Both reporting and reactions are configurable. BIDS provides two categories of non-reporting reactions.

**Termination Actions:** These actions result in the termination of a process or processes. For example, terminate the user's login session.

**Blocking Actions:** These actions result in some type of extended prevention. For example, block incoming connections from a specified host.

Although the basic reaction and reporting capabilities are the same for the two versions of BIDS, some aspects of the implementations differ. All functions, including reaction and reporting, were implemented in user space for BIDS v1. In BIDS v2, the reporting is implemented in user space but some reactions are implemented in kernel space.

The reporting functionality was moved out of the kernel to reduce kernel overhead. Those reactions that involve terminating specific user processes (e.g., terminating the user's login session) are implemented in the kernel to take advantage of the inability of the attacking process to proceed until the intercepted system call returns.

## III. BIDS v2 IMPLEMENTATION

### A. Generic Software Wrapper Toolkit (GSWTK)

Several changes were made to the GSWTK to support BIDS. There are three categories of changes: i) updates to support newer versions of RedHat Linux, ii) bug fixes, and iii) extensions for BIDS v2.

GSWTK 1.6.1 supported RedHat 6.0, the Linux 2.2 kernel, and gcc 2.x. We needed it to work with RedHat 8.0, the Linux 2.4 kernel, and gcc 3.x. The most significant change to support newer versions of the RedHat Linux OS was to rewrite some of the GSWTK system call wrapper initialization functions. The original version of GSWTK used an algorithm that parsed through the assembly language code in the kernel, within a bounded distance from a function entry point, to determine where a replacement should be made. In certain cases, this technique required only two bytes of information to make a match. In a dense instruction set like that of the INTEL 386, a two-byte sequence is insufficient to make a robust match. We modified the replacement algorithm to use the Linux `System.Map` file to acquire the four-byte address of the specified function and then search for the call to that address, a five-byte sequence, within a bounded distance of the beginning of a function. This change resulted in code that was much more stable across Linux kernel versions. Another advantage of using the `System.Map` file is that internal kernel functions no longer need to be exported for the GSWTK to obtain their addresses.

Another fairly major change to the GSWTK, this time in the category of bug fixes, was to be able to support system calls that were nested in the kernel. In Linux, this situation

occurs when it is necessary to load a module for supporting a protocol family for a socket call. A kernel thread is spawned to load the module, resulting in a clone system call nested inside a socket system call. This situation caused a problem in the GSWTK system call interception code since the return information for the socket system call was overwritten by the return information for the clone system call. The solution was to add support for a limited system call stack. The stack depth is restricted so the increase in size of each wrapper is minimal. Any nesting beyond that limit results in a system call error. The system call counter was changed to represent the depth of system call nesting only; it no longer reflects the number of ancestors of that process.

Finally, some extensions were added to the GSWTK to allow more control for the reactions, to provide additional information for the algorithms, and to enable additional control over the wrappers used in BIDS v2.

### B. Analysis and Detection

The primary difference between BIDS v1 and BIDS v2 is the use of system call wrappers. In BIDS v1, the analysis could be treated as a data flow where the input was audit records and the output was intrusion detection alerts. In BIDS v2 each system call must be dealt with individually. This per system call implementation of BIDS results in a distribution of the processing code across system calls.

Because the BIDS algorithms are implemented as wrappers around the kernel system calls, it is important to eliminate false positives without introducing false negatives. No alert should be raised if the system call will fail anyway, for example, if the program to be executed does not exist. We also do not want to prevent valid actions from occurring. We accomplished this objective by adding code to test for the most common failure conditions for the file access system calls.

One challenge was to properly handle processes that are spawned by the kernel with ownership different from that of the original process. This situation can arise only when the thread is spawned and `exec` is called in the kernel, as for module loading. Such processes do not go through any authentication mechanism and thus can generate false alarms. Since the system call counter increments when a new kernel thread is created, if the system call counter is non-zero and the process is a recognized one, the process can be treated as having been authenticated.

Since the BIDS v2 algorithms are implemented in the kernel as part of selected system calls, it is important to try

to keep both the processing and memory overhead low. As one means to that end we reduced the BIDS process database entry by 44% from its size in BIDS v1. In BIDS v2 only derived information is stored; information that can be obtained via kernel data structures is omitted.

The BIDS v2 database is updated whenever processes and connections are created or terminated. Initially, the process database was updated by adding wrappers for the fork family of functions and `exit`. However, this strategy did not work for `vfork`, which doesn't complete until the `vfork'd` process completes, and for cases where the process is terminated by a signal. The GSWTK provides a `wr_duplicate` wrapper function, that is called whenever a process is created, and a `wr_deactivate` wrapper function, that is called whenever a process is terminated. Using these functions to update the process database resulted in full coverage of process creation and termination events.

Another area we looked at was reducing the amount of space required by the database of persistent objects in BIDS v1. In BIDS v1, the entries are stored as a list of strings. In BIDS v2 we use a trie data structure. A trie is useful for dictionary style lookups where the beginnings of different word strings may be the same, as is the case with path names in Unix. In a trie each character is a node. A word is represented as a sequence of nodes connected by edges and terminated by `nil`. A node with more than one child occurs at the point that words with common beginnings diverge. When two words differ in the last letter alone, the number of nodes that need to be stored will be  $n+2$  rather than  $2n$ . For example, the words *anyone* and *any* can be stored as shown in Figure 2, where `/0` is the terminal character.

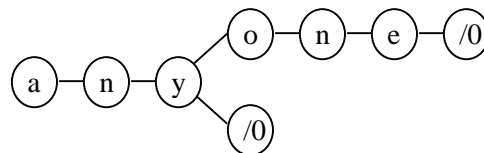


Figure 3. Trie for the words *any* and *anyone*

The BIDS trie was implemented such that directory names are terminated with a slash (`/`) so they can be distinguished from file names. By specially designating directories it is possible to protect all of the files in a directory as well as the directory itself. Using a trie pays off for long path names where the majority of the name may be shared by two files. There is a significant payoff even when the shared path is short. For example, listing all 91 files in `/bin` as complete paths would require 1093

bytes. Listing the same files in a trie requires at most 643 bytes. Using a directory and listing only files with special constraints, as we do configuring BIDS v2, requires 126 bytes, 12% of the original size.

Finally, the BIDS v2 DOM algorithm provides better coverage than the one in BIDS v1 and produces only a DOM alert, not both a POM and DOM alert as in BIDS v1.

### C. Reaction and Reporting

The GSWTK provides a mechanism for communicating from kernel space to user space. We utilized that mechanism for communicating both alerts and reaction directives to the BIDS user space response process.

To support the BIDS v2 reaction mechanism within the kernel, information was added to the process database for tracking process ancestry. In Unix systems, if the parent process terminates, the child process is reparented to the init process. This type of reparenting often occurs during the process chain invoked when a user logs in to a host, either remotely or locally. In order to terminate the user connection from within a session process it is necessary to bridge the process gaps. We do that by maintaining an ancestor id for every process in the BIDS process database. Whenever a process is created, the ancestor id is set to the process id of the parent process. Whenever a process terminates, BIDS finds the children of that process and reassigns the ancestor id of the children to the ancestor id of the terminating process. Thus, when it is necessary to terminate a login session, it is possible to chain backwards from the process to the login process associated with it.

## IV. EVALUATION

To evaluate the ability of BIDS v2 to prevent attacks, we ran a suite of tests designed to exercise the various detection algorithms. The tests use custom programs to implement an attack for each violation type. Most of the tests use a setuid program we engineered to overflow a buffer and execute a script tailored to exercise a specific violation. This suite of tests was developed for exercising BIDS v1; it was extended for BIDS v2 to test all of the system calls monitored by BIDS v2. BIDS v2 successfully prevented all 19 attacks in the test suite. We verified that the same attacks succeeded when BIDS v2 was not installed on the system.

Initially, we had some problems with false alarms. The false alarms fell into three categories: those associated with errors in the wrapper code, those associated with shortcomings in GSWTK, and those related to use of

setuid programs. For the first two types of false alarms, we made adjustments to the relevant code. For the third type, we adjusted the configuration to allow those programs to perform the required activities. When all the tests were then rerun, the attacks were prevented and no false alarms were generated.

BIDS v2 was installed on the FBCB2 platform at Northrop Grumman Mission Systems and run as part of the regular system. Once the configuration was adjusted for that environment, no false alarms were reported. To date there have been no red-teaming efforts against BIDS v2.

For the performance analysis we wanted to evaluate both the per-system-call overhead for representative system calls and the overhead for selected applications. For the system calls we selected *fork*, *execve*, *creat*, *open*, and *chmod*. For the applications we evaluated login processing and the *find* command. All tests were repeated ten times and the average of the ten runs was taken. Because an accurate time measurement could not be obtained for a single wrapped system call, the tests for the wrapped system calls involved timing a loop containing the system call of interest. Since the loops contained other system calls (e.g., *printf/write*), the loops were timed with and without the system call of interest. The results that follow represent the difference between the average time for the loops with the system call of interest and the average time for the same loops without the system call of interest.

**fork:** The fork system function was called inside a loop; each child was exited immediately after creation. The children were reaped when the loop exited. The following overhead is associated with the fork system call: the GSWTK creates a data structure for the new process and BIDS v2 adds a new entry to the process table. No intrusion analysis is performed. The use of a wrapper and the GSWTK database resulted in a factor 1.6 increase in the amount of system time over GSWTK with no wrappers.

**execve:** The *execv* system function was called inside a loop using the name of a non-existent program. The loop also contained two *fprintfs*, one to disk and the other to stdout. The following overhead is associated with the *execve* system call: the GSWTK resets some fields in the wrapper data structure and BIDS v2 tests for unauthorized creation of shell processes. The use of BIDS resulted in a factor 3.2 increase in the amount of system time over GSWTK with no wrappers. The overhead for GSWTK with no wrappers was also a factor of 3.2 over the time required when GSWTK was not loaded.

**chmod, creat, and open:** Each of these functions was called inside a separate loop. Every loop contained a call to `printf`. The `creat` and `open` loops also contained a call to `unlink`. Finally, the loop for `open` also contained a `creat` call, so all opens would be on an existing file. For all of these system calls, GSWTK does no processing beyond wrapping the `chmod`, `creat`, and `open` system calls. All of the calls have a pre-system-call wrapper. In addition, `open` has a post-system-call wrapper. BIDS v2 tests for common errors in the system calls and, if the file is listed in the POM list, tests the file for unauthorized access. For `open`, the access authorization test is performed in both the pre and post wrappers. There was significant overhead for all of the calls: `chmod`, a factor of 14.4; `creat`, a factor of 3.6; and `open`, a factor of 16.3. There was little overhead for GSWTK without any wrappers installed.

**login:** For the login test, the time to login from entering the username and password into the Gnome login dialog box to the appearance of a terminal window was timed. This test exercised the process creation and file access system call wrappers. The test was performed using a stopwatch, so the times are approximate. We saw a factor 0.2 increase, from 6.5 seconds to 7.8 seconds, over running without the GSWTK loaded.

**find:** The find test used the command:

```
% time find / > /dev/null 2>&1
```

The `find` command exercises the wrappers for process creation and accessing files. The overhead associated with running with BIDS was a factor of 4.6 over that of running with no GSWTK loaded.

There is noticeable overhead added by the BIDS wrapper. It may be possible to make modifications to the wrapper code to reduce the overhead somewhat. Even though the overhead seems high for individual system calls, since a process spends most of its time in user space, the overhead is not readily noticeable for applications that don't perform a large number of file operations. Despite that, the possibility of further reducing the processing overhead should be explored in future work.

Another issue we found with using wrapper technology is that version changes to the kernel code usually required some adjustments to the GSWTK. One example of that is when the arguments to a kernel function changed.

## V. RELATED WORK

There have been a number of intrusion prevention systems, some commercial, that use system call interception. Most

of these systems use a policy database to configure the system.

St. Jude is an open source Linux Loadable Kernel Module (LKM) that wraps the functions in the Linux system call table [11]. It protects the system from unauthorized root transitions by implementing a restriction list based on a rule database listing valid transitions. To generate the rule database, St. Jude is run in learning mode.

There are several commercial products [12] [13] [14] that use a policy database to determine whether an action should be allowed or prevented. They also produce alert information. Cisco Security Agent requires information about the application state to reduce the number of false positives. STAT Neutralizer supports configuration of actions to be taken on an alert.

The Linux Intrusion Detection System (LIDS) enforces a configurable access control list [15]. It is implemented as a kernel patch, rather than as an LKM. Once LIDS is loaded, no other LKMs can be loaded. In addition to restricting file, disk, port, and memory access, LIDS also provides the capability to hide specified files and processes and prevent specified processes from being killed, a capability that would be useful in BIDS.

Fnord is a LKM to protect the system against malicious LKMs [16]. It prevents any modules other than itself, even those loaded before it, from replacing entries in the system call table. Fnord is also able to hide itself and other programs and files. It has a number of self-protection and stealth features that would be useful in the GSWTK.

There are a couple of projects that use the GSWTK to implement intrusion detection systems [17][18]. The focus of that work has been on the automated responses to intrusions rather than on intrusion prevention. In the software decoy project, the wrapper uses a simple buffer overflow test. In SHIM, the intrusion detection algorithm determines whether a system call is legal or illegal based on a specification. To reduce system overhead, the GSWTK wrapper reports on system calls of interest but the actual classification is performed by SHIM.

Security-Enhanced Linux (SELinux) provides a mandatory access control mechanism that is built into the Linux kernel [19]. Access control lists are used to configure the policy for role-based access control of files and services. SELinux controls both access to system objects and transitions from one role (user or group) to another. It is implemented in conjunction with the Linux Security Module (LSM). The LSM provides support for security-

based LKMs [20]. If properly configured, SELinux will provide the same protection as BIDS v2. However, it will not provide alert information or the reaction capability provided by BIDS v2. SELinux has stabilized and been incorporated into the released versions of the Linux kernel. At the time that BIDS v2 was implemented, SELinux and LSM were still under development and were not integrated into the kernel used for the FBCB2 and so were not a viable option.

## VI. SUMMARY

We have presented version 2 of BIDS, an intrusion detection system developed for use in the TI. BIDS v2 is effective at detecting and preventing attacks without false alarms. It can be configured to provide more extensive protection in situations where automatic reactions are more desired, such as where systems are forwardly deployed, or a standard level of protection in situations where system administration is more easily supplied, such as where systems are further back in the TI. Although a number of techniques were used to reduce the BIDS v2 overhead, additional work is needed in that area. Despite the system call overhead, a performance impact is not readily noticeable at the system level when the applications are not performing a large number of file opens. BIDS v2 has been successfully integrated with FBCB2 v7.0.3.

## VII. ACKNOWLEDGEMENTS

We would like to thank U.S. Army CECOM for their helpful comments. We would also like to thank Bill Muller, Allan Ratih, Ray Roux, and Hiram Thomas at Northrop Grumman Mission Systems for the screen shot and their inputs.

## VIII. REFERENCES

- [1] G. Sheftick, "Army's 'digitized division' wages first combat," in *ArmyLINK News*, 2003.
- [2] "Tactical Internet," Federation of Am. Sci., 2003, <http://www.fas.org/man/doc-101/sys/land/internet-t.htm>.
- [3] D. Denning, "An Intrusion-Detection Model," *IEEE Trans. on Software Engineering*, vol. 13, 1987.
- [4] R. K. Cunningham, D. A. Kassay, and C. D. McLain, "Battlefield Intrusion Detection System," presented at MILCOM 2003, Boston, MA, USA, 2003.
- [5] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel, 2<sup>nd</sup> edition*. Cambridge, MA: O'Reilly & Associates, Inc. 2003.
- [6] C. Ko, T. Fraser, L. Badger, and D. Kilpatrick, "Detecting and Countering System Intrusions Using Software Wrappers," presented at 9<sup>th</sup> USENIX Security Symposium, Denver, CO, USA, 2000.
- [7] Sun Microsystems, Inc., *SunSHEILD Basic Security Module Guide*, for Solaris Release 8, 2001.
- [8] InterSect Alliance Pty Ltd. SNARE. <http://www.intersectalliance.com/projects/Snare/index.html>
- [9] Rik Faith, Light-weight Auditing Framework, <http://seclists.org/lists/linux-kernel/2004/Mar/0159.html>
- [10] J. Rabek, R. Cunningham, and R. Khazan, "Detecting Privilege-Escalating Executable Exploits," presented at Workshop on Data Mining for Computer Security, Melbourne, FL, USA, 2003.
- [11] T. Lawless, "St. Jude the Model," June 2002, <http://sourceforge.net/projects/stjude/StJudeModel-1.1.pdf>.
- [12] Cisco Systems Inc., "Securing Network Endpoints Without Signatures: A Policy-Based Approach to Host Intrusion Protection," 2003. [http://www.okena.com/application/pdf/en/US/guest/products/ps5057/c1244/cdcont\\_0900aec800ae55e.pdf](http://www.okena.com/application/pdf/en/US/guest/products/ps5057/c1244/cdcont_0900aec800ae55e.pdf)
- [13] R. Henning and R. Caliar, "Behavior-Based Intrusion Prevention: Can Enterprises Afford to Be Without It?," 2003. <http://www.statonline.haris.com/technologies/whitepapers/Paper.Position.Behavior-Based%20Intrusion%20Prevention%202-10-03.pdf>
- [14] Enterscept Security Technologies, "System Call Interception," 2001. [http://www.nai.com/us/products/mcafee/product\\_lit.htm](http://www.nai.com/us/products/mcafee/product_lit.htm)
- [15] H. Xie, The LIDS Project, <http://www.lids.org/>.
- [16] E. Brandwine and T. McDermid, "Fnord: A Loadable Kernel Module for Defense and Honeypots," presented at Blackhat 2003, Las Vegas, NV, USA, 2003.
- [17] J. B. Michael, G. Fragkos, and M. Auguston, "An Experiment in Software Decoy Design: Intrusion Detection and Countermeasures via System Call Instrumentation," In *Proceedings of IFIP 18<sup>th</sup> International Security Conference*, Kluwer Academic Publishers (Athens, Greece, May 2003).
- [18] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt, "Using Specification-Based Intrusion Detection for Automated Response," in *Recent Advances in Intrusion Detection*. Springer-Verlag, 2003.
- [19] P. Loscocco and S. Smalley, "Meeting Critical Security Objectives with Security-Enhanced Linux," presented at USENIX Linux Kernel Developers Summit, Ottawa, Canada, 2002.
- [20] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux Security Modules: General Security Support for the Linux Kernel," 11<sup>th</sup> USENIX Security Symposium, San Francisco, CA, 2002.