CERIAS Tech Report 2000-08

Data collection mechanisms for intrusion detection systems

Eugene Spafford, Diego Zamboni
Center for Education and Research in
Information Assurance and Security
Purdue University, West Lafayette, IN 47907

Data collection mechanisms for intrusion detection systems[*]

Eugene Spafford Diego Zamboni

Center for Education and Research in Information Assurance and Security
1315 Recitation Building
Purdue University
West Lafayette, IN 47907-1315

CERIAS Technical Report 2000-08

June 2, 2000

Abstract

Drawing from the experience obtained during the development and testing of a distributed intrusion detection system, we reflect on the data collection needs of intrusion detection systems, and on the limitations that are faced when using the data collection mechanisms built into most operating systems. We claim that it is best for an intrusion detection system to be able to collect its data by looking directly at the operations of the host, instead of indirectly through audit trails or network packets. Furthermore, for collecting data in an efficient, reliable and complete fashion, incorporation of monitoring mechanisms in the source code of the operating system and its applications is needed.

1 Introduction

In the last fifteen years, since the publication of the first widely known works about intrusion detection [2, 3], a large number of intrusion detection systems have been developed, both as research prototypes and as commercial products.

One of the common classifications of intrusion detection systems is that of host-based and network-based intrusion detection systems [8]. Host-based systems base their decisions on information obtained from a single host, while network-based systems obtain data by monitoring the traffic in the network to which the hosts are connected.

The classification of intrusion detection systems as host-based or network-based usually refers to the way data is collected by the intrusion detection system, and not to how or where it is processed. To make this distinction clear, we use the terms *host-based data collection* and *network-based data collection*.

With respect to where and how data is processed by the intrusion detection system, we classify intrusion detection systems into distributed and centralized. A *distributed intrusion detection system* is one where data is collected and analyzed in multiple hosts, as opposed to a *centralized intrusion detection system*, in which data may be collected in a distributed fashion, but is processed centrally. Both distributed and centralized intrusion detection systems may use host- or network-based data collection methods, or a combination of them.

The study of the data collection mechanisms is important because the detection performed by an intrusion detection system can only be as good (in terms of accuracy, reliability and efficiency) as the data on which it bases its decisions. If the data is acquired with a significant delay, detection could be performed too late to be useful. If the data is incomplete, detection abilities could be degraded. And if the data is incorrect (due to error or to the actions of an intruder), the intrusion detection system could stop detecting certain intrusions, giving its users a false sense of security. Unfortunately, these problems have been identified in existing products. After examining the needs of different misuse detection systems and the data provided by different operating systems, Price concluded that "the audit data supplied by conventional operating systems lack content useful for misuse detection." [10, p. 107]

In this paper we describe different ways of classifying data collection mechanisms for intrusion detection, and discuss the advantages and disadvantages of each one of them according to our experiences in the design and implementation of the AAFID distributed intrusion detection system.

[*]Portions of this work were supported by sponsors of CERIAS

2 Host-based and network-based data collection

Most of the intrusions that existing intrusion detection systems can detect are caused by actions performed in a host: executing a command, accessing a service and providing it improper data, etc. The attacks act on the end host, although they may occur over a network.

The only attacks that act on the network itself are those that flood the network to its capacity, preventing legitimate packets from flowing. However, we claim that most of these attacks can also be detected at the end hosts. For example, a ping flood could be detected at the ICMP layer in the host by looking for the occurrence of a large number of ECHO_REQUEST packets.

The only case in which network-based data collection could be better suited than host-based data collection is for attacks that flood the network with packets that do not cause any reaction on the hosts (for example, packets destined to a port that is closed on all hosts). However, even in this case the attack could be detected at the end hosts in the low levels of the networking stack.

In general, we think it is better to use host-based data collection, for the following reasons:

- Host-based data collection allows the collection of data that reflect accurately what is happening on the host, instead of trying to guess based on the packets that flow through the network.
- In high-traffic networks, a network monitor could potentially miss packets, whereas properly implemented host monitors can report every single event that occurs on each host.
- Network-based data collection mechanisms are subject to insertion and evasion attacks, as documented by Ptacek and Newsham [11]. These problems do not occur on host-based data collection, because they act on data that the host already has.

In a more general sense, these problems reflect the distinction between direct and indirect data collection, which we describe next.

3 Direct and indirect monitoring

At a more general level, we offer the classification of data collection methods in direct and indirect methods, according to the following definitions:

Direct monitoring: Obtaining data directly from the object that generates it, or to which the data pertains. For example, to perform direct monitoring of the CPU load of a host, we need to get the data directly from the appropriate kernel structures in the host. To perform direct monitoring of accesses to network services provided by the **inetd** daemon, we would need to get the data about those accesses directly from **inetd**.

Indirect monitoring: Obtaining data from a source that reflects the behavior of the object that is being monitored. Using the previous examples, indirect monitoring of the CPU load of a host could be done by reading a log file where the CPU loads get recorded. Indirect monitoring of network service accesses could be done by reading a log file generated by the **inetd** daemon or by an auxiliary program such as TCP-wrappers [12]. Indirect monitoring could also be performed by looking at the network for packets destined to the appropriate ports in the host.

For performing intrusion detection, direct monitoring is better than indirect monitoring, for the following reasons:

- Data on an indirect data source (for example, an audit trail) could potentially be altered by an intruder before the intrusion detection system uses them.
- Some events may not be recorded on an indirect data source. For example, not every action of the **inetd** daemon gets recorded to a log file. Furthermore, an indirect data source may not be able to access internal information on the object being monitored. For example, TCP-Wrappers is not able to examine the internal operations of the **inetd** daemon, only the data that gets passed to it through its external interface.
- With indirect monitoring, the data is generated by mechanisms (for example, the code that writes the audit trail) that have no knowledge of the needs of the intrusion detection system that will be using the data. For this reason, indirect data sources usually carry a high volume of data. For example, Kumar and Spafford [5] mention that a C2-generated audit trail might contain 50K-500K records per user per day. For a modest-size user community, this could amount to hundreds of megabytes of audit data per day, as pointed out by Mounji [7].

  For this reason, when indirect data sources are used, the intrusion detection system has to spend more resources in filtering and reducing the data, even before being able to use them for detection purposes.

On the other hand, a direct monitoring method can obtain just the information it needs, resulting in smaller amounts of data being generated. Additionally, the monitoring components could analyze the data themselves and only produce results when relevant events are detected, therefore practically eliminating the need for storing data, other than for forensic purposes (after-the-fact investigation of the events that took place).

- The previous item results in a lack of scalability when indirect monitoring is used, because as the number of hosts (and monitoring elements within each host) increases, the overhead resulting from filtering data can cause degradation in the performance of the hosts being monitored.

- Indirect data sources usually introduce a delay between the moment the data is produced and when the intrusion detection system can have access to them. Direct monitoring, allows for shorter delays, enabling the intrusion detection system to react in a more timely fashion.

4 Experiences in building a distributed intrusion detection system

During the implementation of the AAFID system [1] we faced decisions regarding the use of direct and indirect monitoring, and even when trying to do direct monitoring, we encountered problems with the specific techniques used to perform it. In this section we relate those experiences and how they made us think about data collection methods for intrusion detection systems.

AAFID is a framework for distributed monitoring of hosts in a network, specifically oriented towards intrusion detection. AAFID uses a hierarchical structure of entities, exemplified in Figure 1. At the lowest level in the hierarchy, AAFID agents perform monitoring functions on a host and report their findings to the higher levels of the hierarchy, where data reduction is performed.

AAFID was designed to use host-based data collection, therefore the agents run in each host and collect data from it. To collect reliable data, we wanted to get it directly from each host.

Audit trails are the most abundant source of data in a Unix system, and are the data source used by most intrusion detection systems. In the first implementation of the AAFID system, most of the agents obtained their data from log files. Unfortunately, audit trails are an indirect data source, and suffer from the drawbacks mentioned in Section 3.

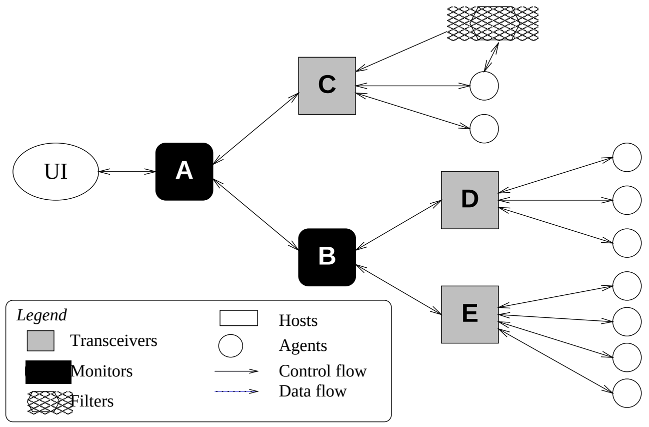

Figure 1: Example of the hierarchical structure of the AAFID system.

To perform direct data collection appropriately, we would have needed operating system support, possibly in the form of hooks to allow insertion of checks at appropriate points in the system kernel and its services. Lacking this support, we decided to implement direct monitoring ourselves using the following mechanisms:

- Separate entities that run continuously, obtaining information and looking for intrusions or notable events. This is the form that most existing AAFID agents have. Some agents obtain information from the system by running commands (such as **ps**, **netstat** or **df**), others by looking at the state of the file system (for example, checking file permissions or contents), others by capturing packets from a network interface (note that this is not necessarily the same as doing network-based monitoring, because in most cases these agents will only capture packets destined to the local host, and not to other hosts), and yet others by looking at audit trails (in some cases, an audit trail is the only place where some information can be obtained).

- Wrapper programs that interact with existing applications or utilities, and that try to observe their behavior by looking at their inputs and outputs.

- Wrapper libraries using library interposition [6]. Using this technique, calls to library functions can be intercepted, monitored, modified or even cancelled by the interposing library. This technique can detect a wide range of attacks, but it is limited because it can only look at the data available as arguments to each call. It cannot have access to any internal data on the calling program.

We classify these techniques of data collection as *external sensors*, because the monitoring components are separate from the programs that are being monitored. The opposite of an external sensor is an *internal sensor*, which is built into the code of the program being monitored.

5 External and internal sensors

External and internal sensors for direct data collection have different strengths and weaknesses, and can be used together in an intrusion detection system. Table 1 lists the advantages and disadvantages of each type of sensor.

From the point of view of software engineering, internal and external sensors present different characteristics in the following aspects:

Introduction of errors: It is potentially easier to introduce errors in the operation of a program through the use of internal sensors, because the code of the program being monitored has to be modified. Errors can also be introduced by external sensors (for example, an agent that consumes and excessive amount of resources, or an interposed library call that incorrectly modifies its arguments). However, we think that most internal sensors can be fairly small pieces of code, allowing them to be extensively checked for errors.

Maintenance: External sensors are easier to maintain because they are separate from the programs they monitor.

Size: Internal sensors can be smaller than external sensors, because they become part of an existing program, thus avoiding the base overhead associated with the creation of a separate process.

Completeness: Internal sensors can access any piece of information in the program they are monitoring, whereas external sensors are limited to externally-available data. For this reason, internal sensors can have more complete information about the behavior of the monitored program. Furthermore, because internal sensors can be placed anywhere in the program they are monitoring, their coverage can be more complete than that of an external sensor, which can only look at the program "from the outside".

Correctness: Because internal sensors have access to more complete data, we expect them to produce more correct results than external sensors, which often have to make educated guesses using the information available.

Our overall comparison of the two types of sensors is that external sensors are better in terms of ease of use and maintainability, whereas internal sensors are superior in terms of monitoring and detection abilities, as well as resilience and host impact. Both types of sensors can be used in an intrusion detection system to take advantage of their strengths according on the specific task each sensor has to accomplish.

In the AAFID system we have experienced first-hand the limitations of indirect monitoring techniques and of external sensors. For this reason, we have started working on using internal sensors for intrusion detection.

6 Implementing internal sensors

Work has started at CERIAS on the implementation of internal sensors on a Unix system. This work is still in its early stages, but the current results are worth mentioning here.

On an OpenBSD [9] system, Kerschbaum [4] has implemented and tested internal sensors for 15 different network attacks, including Land, Teardrop, Ping of Death, Linux blind spoofing, Win Nuke, SYN flooding, port scanning (including full TCP scans, half-open scans and FIN, XMAS and NULL scans) and others. These sensors were implemented by adding only 73 lines of code to the OpenBSD kernel, plus two support files (mostly for data structure manipulation) with a total of 354 lines of code. Most sensors are no longer than 4 lines of code. Also, some sensors are for platforms other than OpenBSD (like Win Nuke, which is a Windows-specific attack), which shows that a single host instrumented with internal sensors can detect attacks for different architectures and operating systems.

The sensors allow detection of attacks in real time, and without the need to execute any additional processes on the system. Although these are only preliminary results, they offer a promising glimpse into the flexibility, efficiency and detection capabilities that internal sensors, implemented with small amounts of code, may offer.

7 Conclusions and related work

We have shown the following (non-exclusive) classifications of data collection techniques for intrusion detection systems:

- *Host-based* and *network-based* techniques. We showed why host-based techniques are better than network-based techniques, except in few specific cases.

	External sensors	**Internal sensors**
Advantages	Easily modified, added or removed from a host. Can be implemented in any programming language that is appropriate for the task.	Minimum delay between the generation of the information and its use. Cannot be easily disabled or modified because they are not separate processes. If implemented correctly, may cause a much smaller performance overhead on the host. Because they are implemented as part of the program they are monitoring, they can access any information that is necessary for their task.
Disadvantages	Can potentially be disabled or modified by an intruder. There is a delay between the generation of the data and their use, because after the data are produced, they have to be made available on an external source before a sensor can access them. Added performance impact because the sensors are separate processes (or additional loaded libraries), running continuously most of the time. Limited in the information they can obtain because they depend on information provided by the system or on information they can gather using existing mechanisms (such as Unix commands).	Much harder to implement, because they need to be incorporated into the program that is going to be monitored. Need to be implemented in the same language as the program they are going to monitor. If designed or implemented incorrectly, can severely harm the performance or the functionality of the program they are part of. Much harder to update or modify.

Table 1: Advantages and disadvantages of external and internal sensors.

- *Direct* and *indirect* data collection techniques. We discussed why direct data collection is more desirable in every case.

- Data collection techniques can be implemented with *external* or *internal* sensors. We presented a comparison of the two techniques, an concluded that although external sensors are easier to implement and use, internal sensors have advantages that make their use worthwhile.

It is important for intrusion detection system designers and implementers to study the data sources they use, to carefully consider the benefits and disadvantages they offer, and whether better techniques could be applied. An intrusion detection system is only as reliable as the data on which it bases its decisions, so it is in our best interest to try to provide our intrusion detection systems with the best possible sources of data.

Finally, we have presented some preliminary results on the implementation of internal sensors on an OpenBSD system. These sensors are capable of detecting a wide variety of attacks with very little code embedded into the OpenBSD kernel. This work is still in its early stages, but the results so far are very exciting and worthy of further investigation.

References

[1] Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, Eugene Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings of the Fourteenth Annual Computer Security Applications Conference*, pages 13–24, December 1998. URL `http://www.cerias.purdue.edu/homes/zamboni/docs/pubs/aafid-acsac98.ps`.

[2] Dorothy E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232, February 1987.

[3] Dorothy E. Denning and Peter G. Neumann. Re-

quirements and Model for IDES – A Real-Time Intrusion Detection System. Technical report, Computer Science Laboratory, SRI International, August 1985.

[4] Florian Kerschbaum. Network attack sensing. Unpublished technical report, May 2000.

[5] Sandeep Kumar and Eugene H. Spafford. A software architecture to support misuse intrusion detection. CSD-TR 95-009, Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398, March 1995. URL `http://www.cerias.purdue.edu/techreports/public/95-04.ps`.

[6] Benjamin A. Kuperman and Eugene H. Spafford. Generation of application level audit data via library interposition. COAST TR 98-17, CERIAS, Purdue University, 1998.

[7] Abdelaziz Mounji. *Languages and Tools for Rule-Based Distributed Intrusion Detection*. D.Sc. thesis, Facultés Universitaires, Notre-Dame de la Paix, Namur (Belgium), September 1997. URL `ftp://ftp.cerias.purdue.edu/pub/doc/intrusion_detection/mounji_phd_thesis.ps.Z`.

[8] Biswanath Mukherjee, Todd L. Heberlein, and Karl N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, May/June 1994. URL `http://seclab.cs.ucdavis.edu/papers/mhl94.pdf`.

[9] OpenBSD. Web page at `http://www.openbsd.org/`, 1999.

[10] Katherine E. Price. Host-based misuse detection and conventional operating systems' audit data collection. Master's thesis, Purdue University, December 1997. URL `http://www.cerias.purdue.edu/techreports/public/97-15.ps`.

[11] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., January 1998.

[12] Wietse Venema. TCP WRAPPER: Network monitoring, access control and booby traps. In USENIX Association, editor, *UNIX Security III Symposium, September 14–17, 1992. Baltimore, MD*, pages 85–92, Berkeley, CA, USA, September 1992. USENIX.