Towards Survivable Intrusion Detection

Chenxi Wang, John C. Knight
Department of Computer Science
University of Virginia
Charlottesville, VA 22903, USA
{cw2e | knight @ cs.virginia.edu}

1. Introduction

Successful Intrusion Detection (ID) is important to ensuring the survivability of a system—one aspect of survivability is to offer continued services in the event of malicious attacks. It is therefore important that any Intrusion Detection System (IDS) employed by the survivability mechanism be survivable itself.

An intruder may target the ID scheme first in order to facilitate further malicious activities. For an intrusion-detection system that is operating in real time, an intruder could be exposed if they were simply to disable the IDS, a denial-of-service attack. A *corrupted* intrusion-detection system, however, might report erroneous information or fail to identify an attack. The consequences could be more serious if intrusion detection were conducted on a network-wide scale, as in the case of network surveillance demanded by system survivability. In such an environment, detection of coordinated attacks relies on information supplied by local intrusion-detection entities distributed across the network [10][13]. An attack targeting a selected set of intrusion-detection components could render the entire network intrusion-detection system ineffective.

This paper identifies vulnerabilities for generic IDSs, and argues that protection of the IDS itself must be dealt with before it can be relied upon to provide the security that is expected. We also outline possible solution techniques and implementation strategies.

2. Dissecting the problem

Figure 1 shows a model that is used in many ID schemes [2][7][10][11]. In this model, event information is collected, preprocessed, and then subjected to intrusion analysis. The results of the analysis are made available via a report facility. The components of the system that perform the analysis and reporting execute typically on dedicated hosts whose security can be enhanced with traditional techniques. Data collection, however, is often carried out, of necessity, within the perimeter of the monitored system itself.

An intruder could disrupt the ID function either by meddling with the raw event data or by direct tampering with the ID components. Ptacek and Newham illustrated examples in which an intruder evades network intrusion detection by manipulating network traffic [9].

Similar attacks can occur for host-based IDSs – an intruder might overwrite the audit logs or interleave malicious activities with normal actions to foil a signature-based IDS.

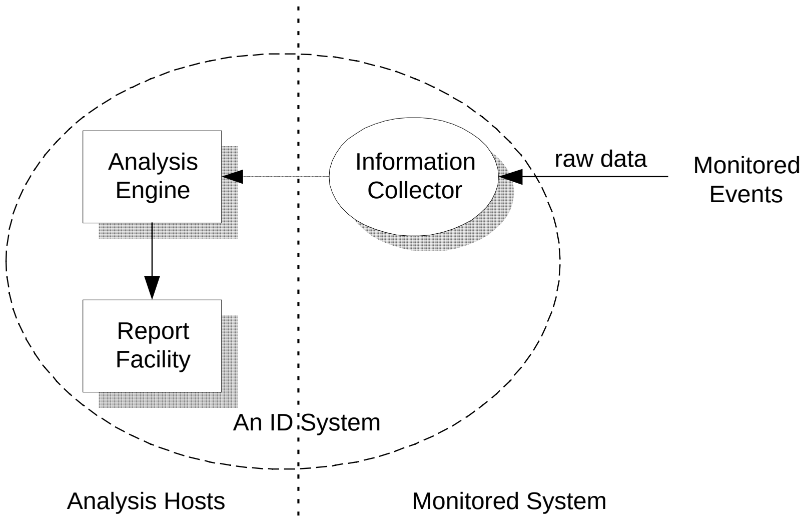

Figure 1: A Generic Model of Intrusion Detection Systems

The easiest target for tampering attacks is the collection of information, the responsibility of the *information collector*[1]. The information collection process (a.k.a. auditing) occurs within the monitored system, and is therefore subject to the same threats as the monitored system. Little research has been undertaken to develop techniques for securing the auditing components or ensuring the authenticity of the data that is being collected. The assumption seems to be that the intrusion detection mechanism should detect the tampering attempt before itself or its data is compromised. This assumption is unfounded and misleading for the following reasons:

- Clandestine users operating at a level below that which auditing occurs can elude the notice of auditing [1][6]. Attacks targeting the IDS may originate at these lower levels, bypass the detection mechanism, and ultimately corrupt the ID data or the mechanism itself.

- It is not always possible to discern suspicious behavior from normal patterns. An anomaly-detection mechanism can be foiled if one can disguise malicious actions as normal activities.

For these reasons, it is possible for an intruder to bypass the ID mechanism and aim the threat directly at the mechanism itself. Protection of the IDS is therefore fundamental to dependable intrusion detection. In the next section, we elaborate on several strategies to enhance the security of ID systems.

[1] The analysis and reporting components can be specially protected on separate hosts, and for the purpose of this discussion, they are assumed to be trusted.

3. Solution Strategies and Implementations

3.1. Diversity Considered Useful

Diversity is an important engineering principle for building robust systems. We argue that the appropriate use of diversity can help to yield sophisticated and tamper resistant ID systems. Two forms of diversity are of special interest:

• **Spatial Diversity:** The use of spatial diversity helps to reduce replicated flaws. When diverse algorithms or implementations are used in a distributed IDS, a widespread class attack exploiting similar flaws will likely be defeated.

• **Temporal Diversity:** Temporal diversity introduces dynamic variations in attributes that will otherwise remain static. Attacks based on analyzing static attributes can therefore be thwarted. For example, if monitoring consists of reading audit logs only, an intruder might overwrite the audit trails to evade detection. A solution to this problem is to change the way data is gathered—periodically, the IDS can switch from reading audit trails to an active memory scan or a real-time signature checking on the invariant part of the program.

Diversity can be incorporated in the implementation as well as design of the software component (e.g. different detection algorithms). The variations can be instituted in an installation specific fashion, and a new installation can occur at random intervals. This forces attacks to be specifically tailored for each installation, and, if it is difficult to predict which installation is to be used next, the resources required to mount an attack would rise significantly.

3.2. Obstruction of Program Analysis

Targeted tampering of software requires knowledge about the program, and one way this knowledge can be acquired is through program analysis. Two forms of program analysis are relevant:

• **Static Analysi**s: This refers to examining and extracting information from a static program image. For example, by scanning the program image, one might be able to discern where buffers holding log records are located and what their formats are.

• **Dynamic Analysi**s: Simulation, black-box analysis and execution profiling, are three examples of dynamic analysis. Dynamic analysis provides run-time information from which sufficient insight might be gained to allow legitimate behavior to be mimicked.

Our research focuses on defeating static analysis[2]. A sophisticated intruder might copy the executable form of the IDS components and analyze them statically to determine, for example, what data is being collected and how it is forwarded to the analysis component.

[2] The discussion on dynamic analysis is omitted here for space limitations.

Our approach is to defeat static analysis by creating executable forms of the IDS components that either inhibit static analysis techniques or make them prohibitively and expensive to carry out. We discuss several techniques to obstruct program analysis together with a compiler-based implementation that transforms source programs into executable forms which incorporate the obstruction techniques.

a) **Control-flow Transformations:** Program control flow provides information for further analysis. By adding nonfunctional code and reorganizing control constructs, we can transform the control flow to an arbitrarily complex form. Substituting direct branches with data-dependent indirect branches, and basing function calls on function pointers for example, severely inhibits static control-flow analysis [3][12].

b) **Data-flow Transformations:** Knowledge of data usage provides information that can facilitate tampering attempts. Data-flow transformations can be used to obstruct data usage analysis. Systematic introduction of aliases, for example, impedes such analysis – most data-flow problems hinge on alias detection, and precise alias analysis is known to be NP hard [4][5][8].

These strategies can be implemented as code transformations during compilation. Our implementation consists of source and intermediate-level transformations, and each compilation performs a unique set of transformations determined by a random seed. The end result is a large array of diverse program versions that are difficult to analyze.

Deploying these diverse versions of ID software, along with the notion of temporal replacement, affords powerful protections against all but the most well facilitated adversaries.

We are presently engaged in an experimental study of the practical efficacy of these techniques. We are assessing the difficulty of analyzing transformed programs using static analysis techniques. Preliminary results indicate that in many cases, static analysis using existing tools is completely defeated [12].

4. References

[1] J. P. Anderson, “Computer Security Threat Monitoring and Surveillance”, *Technical Report, James P. Anderson C*o. Fort Washington, Pennsylvania, April 1980.

[2] Staniford-Chen, S., Tung, B., and Schnackenberg, D. “The Common Intrusion Detection Framework (CIDF)”. Information Survivability Workshop, Orlando FL, October 1998.

[3] M. Hecht. “Flow Analysis of Computer Programs”, Elsevier North-Holland, New York, 1977.

[4] W. Landi. “Interprocedural Aliasing in the Presence of Pointers”. *Ph.D. Dissertation, Rugters Universit*y, 1992.

[5] W. Landi. “Undecidability of Alias Analysis”. *ACM Letters on Programming Languages and Systems,* Vol. 1, No. 4, December 1992, pp 323-33*7*.

[6] T. Lunt, “A Survey of Intrusion Detection Techniques”, *Computers & Securit*y, Vol 12, 1993, pp 405-418.

[7] T. Lunt, “Detecting Intruders in Computer Systems”, In the proceeding of Conference on Auditing and Computer Technology. Canada 1993.

[8] E. Myers, “A Precise Interprocedural Data Flow Algorithm” In the Conference Record of the 8th Annual ACM Symposium on Principles of Programming Languages. 1981

[9] T. Ptacek, T. Newsham, “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection”, *Technical Report, Secure Networks, In*c. January, 1998.

[10] P. Porras, P. Neumann, “EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances”. In *proceedings of the 1997 National Information Systems Security Conferenc*e. Baltimore, 1997.

[11] B. Silken, A. Jones, “Application Intrusion Detection”, *Technical Report, CS-2000-1*7. Department of Computer Science, University of Virginia.

[12] C. Wang, J. Hill, J. Knight, J. Davidson “Software Tamper Resistance: Obstructing Static Analysis of Programs", *CS Technical Report, CS-2000-1*2. Department of Computer Science, University of Virginia.

[13] G. Vigna and R. Kemmerer, "NetSTAT: A Network-based Intrusion Detection System", *Journal of Computer Securit*y, 7(1), IOS Press, 1999.