Attack Languages

Giovanni Vigna, Steven Eckmann, and Richard Kemmerer
Reliable Software Group
Department of Computer Science
University of California
Santa Barbara, CA 93106

Intrusion detection systems (IDSs) play an important role in achieving survivability of information systems. The goal of intrusion detection systems is to identify malicious behavior at different levels of granularity. The data provided by IDSs are the basis for procedures aimed at assuring that an occurring attack will not prevent a system from delivering the expected functionalities.

Attacks against a system have a "manifestation" in terms of events. These events can be of different nature and level of granularity. For example, they may be represented by network packets, operating system calls, audit records produced by the operating system auditing facilities, or log messages produced by applications.

The goal of IDSs is to analyze one or more event streams and identify manifestations of attacks. Historically detection has been achieved following two different approaches: *anomaly detection* and *misuse detection*. Anomaly detection relies on models of the "normal" behavior of a computer system. The models may focus on the users, on the applications, or on the network. Behavior profiles may be built by performing statistical analysis on historical data [7] or by using rule-based approaches to specify behavior patterns (e.g., [8, 16]). Anomaly detection compares actual usage patterns against the established profiles to identify abnormal patterns of activity. Misuse detection systems take a complementary approach. Misuse detection tools are equipped with a number of attack descriptions. These descriptions (or "signatures") are matched against the stream of audit data looking for evidence that the modeled attack is occurring.

When an IDS identifies malicious behavior, it performs a response, which is a series of actions that have to be carried out to react to the attack. For example, a response may include reporting to a human Security Officer, sending a message to an application, or performing some kind of countermeasure to protect the system from the attack (e.g., firewall reconfiguration).

Intrusion detection systems operate in different environments and in different domains. They have to be able to recognize different types of malicious behavior and, in addition, the response to malicious behavior may be different according to the level of protection and the survivability requirements of the system being protected. As a consequence the intrusion detection tasks are supported by a number of languages, often generically referred to as "attack languages".

More specifically, attack languages are needed to encode the "manifestations" of an attack into a suitable format, to recognize an attack given a manifestation, and to react to or report an attack. Attack languages are also useful for analyzing the relationships among different attacks in order to identify coordinated attacks against a system. In addition, attack descriptions can be used to describe attack histories/scenarios and they can be used for reproducing attacks, for testing purposes.

Recently there has been much discussion about attack languages in the survivability community. The goal of the discussion was to identify a "common attack language" that could be adopted and shared by the different research groups in the Intrusion Detection community. Such a language would allow easy integration between different systems and would avoid "replication" of research efforts.

The purpose of this paper is to point out that although these discussions claim to be about a "common attack language", the "attack languages" that have been considered and discussed are of different nature and scope. It is our opinion that if the community wants to make progress in this area, then we need to be clear as to what specific type of language is the object of discussion and what the scope and requirements for such a language are.

In this position paper, we classify the existing "attack languages" according to six different language classes: event languages, response languages, reporting languages, correlation languages, exploit languages, and detection languages. In the following sections we consider each of these language classes, give a brief description of their scope and properties, give examples where possible, and discuss the possibility of having a "common" language for the class under consideration.

Event Languages Event languages are used to describe "events." These events are the basic input for security analysis. This class of languages is mainly focused on the specification of data format. In most cases, there is not a formal language definition; rather, there is some natural language description of the format of each event type and a schematic description of the associated structure. There are many examples of event languages with different formats and/or features: BSM audit record specifications [17], tcpdump packets [10], syslog messages [18], etc. A well-defined common event language would be beneficial to the ID community. It would support component and preprocessor reuse, simplify data sharing among different systems, and allow for the merging of different event streams. An example of a proposed standard format is given in [1]. Other general content specification languages, such as XML [19], could also be used.

Response Languages Response languages are used to specify the actions to be taken in reaction to the detection of attack manifestations. Most IDSs currently do not have a well-defined response language. Instead they use library functions written in general-purpose languages (e.g., C or Java) and have limited customizability and extendibility.

A common response language would be useful so that response procedures could be defined once and installed in different intrusion detection systems. In addition, a common language would support dynamic reconfiguration of responses. For example, different types of responses could be activated on the basis of the evolution of an attack.

Reporting Languages One of the possible responses to an attack is the reporting to a human Security Officer or to an application. Reporting languages are used to describe alerts containing information about an attack, such as source of the attack, target of the attack, type of the attack (if known), events that are part of the manifestation, etc. Note that a reporting language could also be used as an event language at a higher level, for example as input to correlators. A common reporting language has been identified as an asset by the ID community and standard reporting language formats have been proposed. Two examples of reporting languages are the Common Intrusion Specification Language (CISL) [5] and the Intrusion Detection Message Exchange Format (IDMEF) [2]. CISL is part of the Common Intrusion Detection Framework (CIDF) [6]. The language is based on the concept Generalized Intrusion Detection Objects, called GIDOs, which, in turn, are specified as S-expressions. IDMEF is a product of the Intrusion Detection Working Group (IDWG). IDMEF is based on XML and it builds on much of the previous CIDF work. The language is currently undergoing the Internet Engineering Task Force (IETF) standardization process.

Correlation Languages Correlation languages are currently the focus of much research in the intrusion detection and response community. These languages rely on the semantically rich alerts provided by different intrusion detection systems and attempt to recognize "the big picture." That is, they specify relationships among attacks to identify coordinated attempts to break the security of an information system and identify the impact on the target system. The application of a correlation language is sometimes the application of existing detection techniques at a higher level of abstraction. For instance, instead of analyzing BSM events and tcpdump events, correlators may use similar techniques to analyze alerts. Examples, of current approaches to the correlation problem are the use of Bayesian networks, as demonstrated in Honeywell's ARGUS system and SRI's EMERALD, event-based reasoning, as demonstrated in UCSB's STATL [4], and rule-based reasoning, as in SRI's P-Best [9].

Exploit Languages Exploit languages are used to describe the steps to be followed to perform an intrusion. They are usually executable general-purpose languages such as C, C++, Perl, Tcl, Bourne Shell, Python, etc. There are also languages that are explicitly designed to support the scripting of attacks. Examples are CASL (Custom Attack Simulation Language) [12] and NASL (Nessus Attack Specification Language) [3]. Both these languages provide language-level support for attack scripting. A common exploit language would be useful to the security community.

For example it would simplify the generation of test cases for intrusion detection systems. In addition, it would support the sharing and understanding of attacks among the members of the security community.

Detection Languages Detection languages are designed to support intrusion detection, and they are usually referred to as "attack languages." These languages provide mechanisms and abstractions for identifying the manifestation of an attack. Well known examples of detection languages are P-Best [9], which is the rule-based component of SRI's Emerald, UCSB's STATL [4], which is used by the STAT Toolset, N-code [14], used by the Network Flight Recorder, the languages used by the Bro [13] and Snort [15] systems, RUSSEL [11], which is the language used by ASAX, and specification languages such as [8] and [16].

Need for a Common Detection Language Approximately a year ago the DARPA community formed an Attack Language Group to coordinate DARPA-sponsored research on "attack languages" and to avoid "replication." This group has met several times over the past year and has also had several teleconferences. It is our opinion that within this group there is confusion concerning the different language classes, the differing goals, and the differing scopes. An unfortunate side effect of this confusion is that the trend towards standardization of a "common attack language" turned out to be an attempt to define a common "detection language".

As we discussed in the previous section, we believe that standardization of event languages, report languages, response languages, and exploit languages would be a good thing. Conversely, we think that the standardization of detection and correlation languages would be detrimental to the intrusion detection community. We feel that there is a need for more exploration into different, complementary approaches. After gaining experience with these approaches, we may then be ready to consider a common detection/correlation language. Community-wide adoption of a single attack detection language now would stifle exploration of new ideas.

A useful approach would be to devise procedures (1) to evaluate the characteristics of existing languages with respect to a set of general requirements and (2) to perform comparisons among languages so that the respective advantages/disadvantages, scope, and characterizing features could be better understood.

The requirements for a detection language include: simplicity, expressiveness, rigor, extensibility, and executability or translatability. A detection language should be as simple as possible; that is, it should provide just the features needed to represent attack scenarios. Although the language should be simple it should be expressive enough to represent any attack scenario that is detectable. The language should also have a rigorous syntax and semantics. Because it is not possible to anticipate all attack scenarios and because some event types, or some elements of the discourse (e.g., IP addresses) may be relevant for one system but not for others, a detection language should also be extensible, and there should be a well-defined way of extending it. Finally, because detection languages are intended to support intrusion detection, it should be possible to incorporate attack descriptions in an ID application for automatic processing of the input event stream. This can be done directly or through an automatic translation process of the attack description into a format supported by an execution run-time.

Comparison among languages could be carried out in different ways. Examples of these efforts could be the analysis of the translatability between different detection languages belonging to similar classes (e.g., signature-based or specification-based) and the use of example scenarios to test the expressiveness of languages. For example, the STAT group at UCSB is working on analyzing the translatability of STATL to and from other signature-based detection languages such as P-BEST, Snort, and N-Code. As an example of the use of example scenarios, we suggest defining a set of abstract scenarios and challenging language developers to write detection specifications or programs for each. For instance, consider the following exercise.

> Design a signature that detects an attack composed of n events of a certain type in a window w. An event is a tuple $\langle From, To, Timestamp, Type \rangle$, where $From$ and To denote the sender and receiver of an event, $Timestamp$ is the generation time for the event, and $Type$ describes the type of event. The events are directed at the same victim, but they could come from different attackers. The attack should be considered "in progress" once the threshold n has been reached, and the attack should be considered over when a timeout t expires after the last notable event. The attack report should contain the following information: victim, attacker(s), start time, and end time.

Similar examples could be proposed to exercise other characteristics of detection languages. By encoding the same abstract scenario in different detection languages it would be possible to understand if a certain attack is detectable and how easy it is to express the relationships among the events involved.

References

[1] M. Bishop. Standard Audit Trail Format. In *Proceedings of the 1995 National Information Systems Security Conference*, pages 136–145, Baltimore, Maryland, October 1995.

[2] D. Curry. Intrusion Detection Message Exchange Format: Extensible Markup Language (XML) Document Type Definition. `draft-ietf-idwg-idmef-xml-01.txt`, July 2000.

[3] R. Deraison. *The Nessus Attack Scripting Language Reference Guide*, 2000. `http://www.nessus.org`.

[4] S. Eckmann, G. Vigna, and R. Kemmerer. STATL. Technical report, UCSB, 2000.

[5] Common Intrusion Detection Framework Working Group. A CISL Tutorial. *http://www.gidos.org/tutorial.html*, 2000.

[6] Common Intrusion Detection Framework Working Group. Common Intrusion Detection Framework Specification. *http://www.gidos.org/*, 2000.

[7] H. S. Javitz and A. Valdes. The NIDES Statistical Component Description and Justification. Technical report, SRI International, Menlo Park, CA, March 1994.

[8] C. Ko, M. Ruschitzka, and K. Levitt. Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 175–187, 1997.

[9] U. Lindqvist and P.A. Porras. Detecting Computer and Network Misuse with the Production-Based Expert System Toolset (P-BEST). In *IEEE Symposium on Security and Privacy*, Oakland, California, May 1999.

[10] S. McCanne, C. Leres, and V. Jacobson. Tcpdump 3.4. Documentation, 1998.

[11] A. Mounji. *Languages and Tools for Rule-Based Distributed Intrusion Detection*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix Namur (Belgium), September 1997.

[12] Secure Networks. *Custom Attack Simulation Language (CASL)*, January 1998.

[13] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, January 1998.

[14] M.J. Ranum, K. Landfield, M. Stolarchuck, M. Sienkiewicz, A. Lambeth, and E. Wall. Implementing a Generalized Tool for Network Monitoring. In *Eleventh Systems Administration Conference (LISA '97)*. USENIX, October 1997.

[15] M. Roesch. *Writing Snort Rules: How To write Snort rules and keep your sanity*. `http://www.snort.org`.

[16] R. Sekar and P. Uppuluri. Synthesizing Fast Intrusion Detection/Prevention Systems from High-Level Specifications. In *Proceedings of the USENIX Security Symposium*, 1999.

[17] Sun Microsystems, Inc. *Installing, Administering, and Using the Basic Security Module*. 2550 Garcia Ave., Mountain View, CA 94043, December 1991.

[18] syslog(3). UNIX documentation.

[19] World Wide Web Consortium (W3C). Extensible Markup Language (XML). W3C Recommendation, February 1998. http://www.w3.org/TR/1998/REC-xml-19980210.