

CARDS: A DISTRIBUTED SYSTEM FOR DETECTING COORDINATED ATTACKS

Jiahai Yang, Peng Ning, X. Sean Wang, and Sushil Jajodia

Center for Secure Information Systems

George Mason University

Fairfax, VA 22030, USA

{yjh, pning, xywang, jajodia}@ise.gmu.edu

Abstract A major research problem in intrusion detection is the efficient Detection of coordinated attacks over large networks. Issues to be resolved include determining what data should be collected, which portion of the data should be analyzed, where the analysis of the data should take place, and how to correlate multi-source information. This paper proposes the architecture of a Coordinated Attack Response & Detection System (CARDS). CARDS uses a signature-based model for resolving these issues. It consists of signature managers, monitors, and directory services. The system collects data in a flexible, distributed manner, and the detection process is decentralized among various monitors and is event-driven. The paper also discusses related implementation issues.

Keywords: computer networks, intrusion detection, misuse detection, network security

1. INTRODUCTION

With the rapidly growing connectivity of the Internet, networked computer systems are fulfilling increasingly vital roles in our modern society. While the Internet has brought great benefits to this society, it has also made critical systems vulnerable to malicious attacks [3]. Coordinated attacks are increasingly popular among hackers; such attacks are difficult to detect and effectively defend.

The conventional approach to secure a computer or network system is to build a protective shield around it (e.g., a firewall). Outsiders who need to access the system must be identified and authenticated [8]. Since such a preventive approach is not sufficient to provide sufficient security for a computer system, intrusion detection techniques are introduced as a second line of defense [2, 8].

Early intrusion detection system (IDS) models were designed to monitor the activities of a single host. Such models include Haystack [12] and SRI's IDIS [5, 7]. Later models accommodated the monitoring of a number of hosts in-

terconnected via a network. Examples include University of California-Davis' Network Security Monitor [4] and DIDS [13]. More recent models, such as UC-Davis' GrIDS [14], UC-Santa Barbara's NetSTAT [15], Purdue's AAFID [1], and SRI's EMERALD [11], pay more attention to intrusion detection for large-scale distributed networks. An important research problem these systems address is how to detect coordinated attacks over large distributed systems (e.g., Mitnick attack [10] and Internet worm incident). These systems also address what data should be collected, where the analysis of these data should be accomplished, and how to correlate multi-source information.

Although significant progress has been achieved by these new systems, additional research is needed to develop more practical systems. In this paper, we propose the architecture of Coordinated Attack Response & Detection System (CARDS), which focuses on detecting coordinated attacks over large-scale, distributed systems. CARDS adopts a scalable, high-level, signature-based intrusion detection approach [9]. The approach uses an audit-independent, structured format to model known attack patterns and represent lower level audit trail or network traffic information. One advantage of CARDS is that multi-source information correlation can be achieved more easily. Another advantage is that the proposed system adopts a decentralized analysis and detection mechanism, so that the single points of failure can be removed. Section 2 gives a brief description of our model. More detailed information can be found in [6, 9].

The rest of the paper is organized as follows. In section 3, we describe the architecture of CARDS. Section 4 describes the approach that CARDS uses to generate and distribute detection tasks and cooperatively detect attacks. We discuss implementation issues in section 5. Section 6 provides conclusions and addresses future work.

2. SIGNATURE-BASED ATTACK MODEL DESCRIPTION

In this section, we briefly describe the signature-based attack model presented in [9]. We formalize the structure of relevant information as a *system view*. We model an attack as a pattern (which we call a *signature*) of events on multiple system views.

SYSTEM VIEW A system view is an interface between a signature and the real system, which provides the event and the state information of a target system. A system view consists of an event schema and a set of (dynamic) predicate names. The event schema specifies the event attributes, each with an associated domain of values. The (dynamic) predicate names are prototypes of Boolean functions that represent relationships among some system entities. For example, we may have a system view (*EvtSchema*, *PredicateSet*) for UNIX hosts, where $EvtSchema = \{subject, action, object\}$ and *PredicateSet*

$= \{same_object [var_time] (file1, file2), same_owner [var_time] (file1, file2)\}$. The elements *subject*, *action*, and *object* in *EvtSchema* are event attributes. The dynamic predicate *same_object* is a Boolean function prototype that should return *true* iff *file1* and *file2* represent the same file object at time *var_time*. The predicate *same_owner* is defined similarly.

An *event* on a system view is a tuple on the event schema with an interval timestamp [*begin_time*, *end_time*], where the tuple consists of the event attribute values and the timestamp represents the interval during which the event occurs. A finite set of events on a system view and the Boolean functions that evaluate the dynamic predicates for the time during which these events occur are collectively called an *event history* on the system view.

SIGNATURE Signatures are event patterns representing intrusive activities that may occur across multiple systems. A signature is specified by a set of events and the constraints that these events must satisfy.

We model a signature as a labeled directed graph. Each node in the graph corresponds to an event on a particular system view and each arc is labeled with a temporal relationship between (the timestamps of) the two nodes (events) involved in the arc. Events matched to the nodes must satisfy certain conditions, which are built into the model by associating a *timed condition* with each node. Assignments of attributes to variables are used to enhance the specification of timed conditions. Furthermore, we distinguish two types of nodes: *positive* and *negative*. Positive nodes represent events (which we call *positive events*) that are necessary for conducting an attack, while negative nodes represent those events (which we call *negative events*) that if they coexist with the positive events, then the positive events do not constitute an attack.

Figure 1 shows the signature of the Mitnick attack described in [10]. Two system views are involved: $DOSAttacks = (EvtSch1, \{ \})$ represents the denial-of-service attacks detected by a network monitor, where *EvtSch1* consists of the attributes *Attack*, *Protocol*, *VictimIP*, and *VictimPort*; $TCPConnView = (EvtSch2, \{Local_IP[var_time](var_IP), Trust[var_time](var_host)\})$ represents the TCP connections on a host, where $EvtSch2 = \{SrcIP, SrcPort, DstIP, DstPort\}$, $Local_IP[var_time](var_IP)$ evaluates to *true* if and only if *var_IP* belongs to the local host, and $Trust [var_time](var_host)$ evaluates to *true* iff the local host trusts *var_host* at time *var_time*. The system view declaration illustrates that *SysView1*, *SysView2*, and *SysView3* are instances of the corresponding views. The signature has three nodes (events). Node *n1* represents a SYN flooding attack against a TCP port on a host, say *A*; node *n2* represents a TCP connection seen on a host, say *B*, which trusts *A*. The timed condition with *n2* says that this connection is from the port being flooded. Both *n1* and *n2* are positive nodes (shown in solid circles), while node *n3* is a negative node (shown in a dashed circle). The labeled arc from *n2* to *n1* says that the event of *n2* should occur during that of *n1*, and the labeled arc from *n3* to *n2* and the timed condition with

$n3$ say that $n2$ and $n3$ are the same connection. This signature describes the Mitnick attack, in which a TCP connection is made from a port being flooded while the host being flooded does not report the same connection.

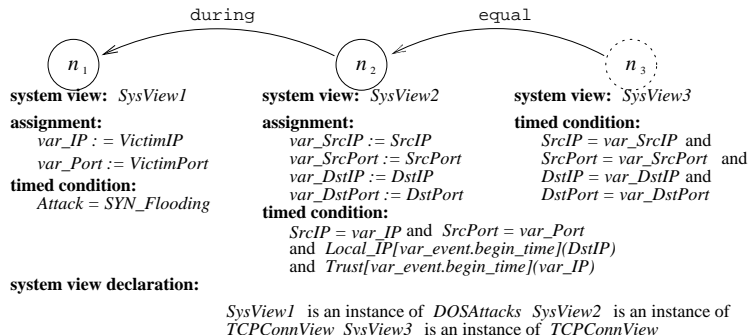


Figure 1 The signature for the mitnick attack

Note that while a signature may represent an attack that can be detected by an intrusion detection system, we can also correlate the result of various intrusion detection systems by writing signatures on the basis of their outcome. The above signature shows such a correlation, in which the SYN flooding attack may be detected by another intrusion detection system.

SPECIFIC SIGNATURE A signature is an event pattern over a set of system views. Such a signature is also called a *generic signature*, as it does not refer to any specific hosts. The signature in Figure 1 is an example of a generic signature. Generic signatures need to be associated with the real systems before the corresponding attack on these systems can be detected. A signature is called a *specific signature* if each system view used by the signature is associated with a particular component that provides information through the system view.

3. ARCHITECTURE DESCRIPTION

CARDS is a distributed intrusion detection system composed of three types of independent but cooperative components: *signature manager*, *monitor*, and *directory service*. Figure 2 shows the architecture of CARDS. In a typical environment, there may be one or more signature managers and one or more monitors. The monitors can be embedded in the monitored system or as a dedicated system separate from the monitored system. Different monitors can cooperate with each other through message passing when they are involved in the detection of one attack.

SIGNATURE MANAGER As shown in figure 2, with the *monitor* configuration information retrieved from the directory service, a signature manager (1) generates specific signatures from generic signatures, (2) decomposes specific

signatures into intrusion detection tasks, and (3) distributes these tasks to the involved monitors.

A detection task is a part of a specific signature that is assigned to a monitor. When a specific signature involves only one monitor, it will have a single detection task. Usually, a signature representing a coordinated attack involves multiple monitors. Such a signature will be decomposed into several detection tasks that the corresponding monitors need to perform.

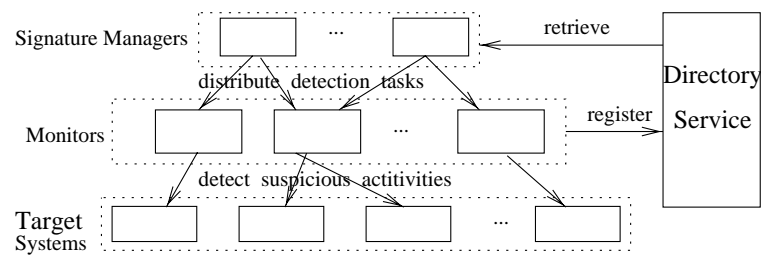


Figure 2 The CARDS architecture

MONITOR Monitors are the components that carry out the intrusion detection tasks. At the beginning of detection, each monitor receives detection tasks from signature managers. During detection, it cooperates with other monitors if some detection tasks are parts of some coordinated attacks.

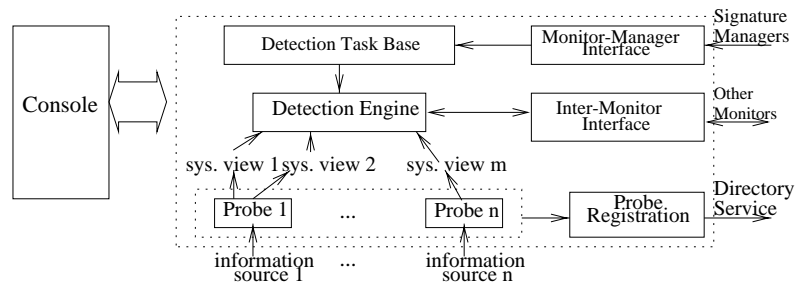


Figure 3 The monitor architecture

Figure 3 shows the architecture of a monitor, which is composed of several *probes*: a *probe registration module*, a *detection engine*, an *inter-monitor interface*, a *detection task base*, a *monitor-manager interface*, and a *console*.

Probes are responsible for collecting information from the target system, filtering and reformatting the information into structures defined by system views, and providing the results to the detection engine. Each probe gets information from one particular source, such as a host audit trail. The system view config-

uration of each probe (i.e., system views that each probe provides) should be put into the directory service via the probe registration module, which is later used by signature managers to generate specific signatures.

The monitor-manager interface receives intrusion detection tasks from the signature manager and stores them in the detection task base.

The detection engine is the core module of the monitor that analyzes the information provided by the probes in terms of the detection tasks. When a detection task belongs to a specific signature involving several monitors, the detection engine then cooperates with the detection engines in the related monitors by passing messages through the inter-monitor interface. The console is the user interface of the monitor. The administrator of a monitor can inspect the status of the monitor and make local configurations through the console.

DIRECTORY SERVICE The directory service (DS) is the information center for providing system-wide information to both signature managers and monitors. Although DSs may be distributed or replicated, they function as a single component of the system. Thus, the signature managers and the monitors are allowed to work in a decentralized and scalable manner and deal with only the components necessary for conducting the designated detection tasks.

The directory service provides two types of information: system view definition and system view configuration. The system view definition specifies the structures and the semantics of the system views. Once a system view is defined, its definition should be placed in the directory service. The system view configuration information specify the system views of the probes. As described above, the probe registration module of the monitors updates this information when a monitor is deployed or reconfigured.

4. COORDINATED ATTACK DETECTION

In this section, we discuss approaches that signature managers use to generate and decompose specific signatures and the procedures that monitors use for cooperatively detecting the coordinated attacks. Before going into the detail, we first introduce the notion of serializable signature.

SERIALIZABLE SIGNATURE For any two nodes n and n' in a given signature, we say n requires n' if the variables in the timed condition associated with n appear in the assignments associated with n' . Intuitively, n requires n' means that node n needs information from node n' through the variable assignments. For example, in the signature shown in figure 1, $n2$ requires $n1$ and $n3$ requires $n2$. We say a signature is *serializable* if there exists a total order of the nodes in the signature such that for each node n , all the nodes that it requires appear before it and all positive nodes appear before negative nodes. For example, the signature shown in figure 1 is serializable, since the total order $n1, n2, n3$ satisfies the above serializable condition. If a signature is

serializable, each node in the signature only needs information from the nodes before it in the corresponding total order. To simplify the discussion, we will only consider the detection of serializable signatures in this paper.

SPECIFIC SIGNATURE GENERATION We say a system view v used by a signature is *associated* with the probe p of the monitor m if p is designated to provide information for the signature through v . In CARDS, a signature is called a specific signature if each system view used by the signature is associated with a probe of a monitor in the system. Thus, the task of specific signature generation is to associate all of the system views used by the signature with the probes of the monitors in the system.

When generating specific signatures from a generic one, the signature manager first looks in the directory service to identify the probes that have the system views used by the generic signature. Then the signature manager derives specific signatures by associating the system views to the probes of the monitors under its control. For example, consider the signature in figure 1. Suppose that a signature manager and three monitors exist, called *Sniffer*, *Megalon* and *Backeast*. If the signature manager learns that the probe *DOSProbe* of *Sniffer* has the system view *DOSAttacks* and both the probe *ServerTCPConn* of *Megalon* and the probe *ClientTCPConn* of *Backeast* have the system view *TCPConnView*, it can generate a specific signature by associating the corresponding system views with these probes. This specific signature then describes the Mitnick attack against the hosts monitored by *Megalon* and *Backeast*.

One generic signature may generate more than one specific signature. For example, a monitor *Outwest* may have the system view *TCPConnView* and the host monitored by *Megalon* trusts the host monitored by *Outwest*. The Mitnick attack may be launched against these two hosts as well; thus, the signature manager should have a specific signature similar to the one above.

Finally, optimization can be used to generate specific signatures. For example, the specific signature that we discussed earlier needs to be generated only if the host monitored by *Megalon* trusts the host monitored by *Outwest*.

SPECIFIC SIGNATURE DECOMPOSITION Having generated the specific signatures, the signature manager should decompose each specific signature into detection tasks for the monitors involved so that they can cooperatively detect these attacks. As coordinated attacks usually cannot be reliably detected at a single location, two or more monitors are often involved in one specific signature. Therefore, specific signature decomposition is necessary.

A given signature is decomposed in two steps. In Step 1, the signature manager arranges the nodes of the signature in an order such that (1) for each node n in the specific signature, all the nodes that n requires appear before n ; (2) as many nodes that belong to the same monitor can be adjacent to each other as possible but condition (1) is still satisfied; and (3) all negative nodes appear

after all positive nodes and yet condition (2) is still satisfied. Such a sequence of nodes is called a *node chain*.

In Step 2, the node chain is partitioned into groups such that each group consists of the nodes whose system views belong to the same monitor. Each group is then assigned to the corresponding monitor and represents the detection task for which the monitor is responsible.

We also get an ordered list of monitors as a byproduct of the specific signature decomposition. We call such a sequence of monitors a *monitor chain* and denote it in the form of $monitor_1 \rightarrow monitor_2 \rightarrow \dots \rightarrow monitor_i^* \rightarrow \dots \rightarrow monitor_n$, where the monitor marked with “*” is the last one having positive nodes. The arrow represents the information flow in the cooperative detection performed by the monitors in the monitor chain. Note that a monitor may appear more than once in a monitor chain.

COOPERATIVE DETECTION Due to space limitations, we only outline the detection procedure in this paper. To facilitate the presentation, we define some terms regarding the monitors in the monitor chain. We call the first monitor in a monitor chain the *beginning monitor*, and the last one in the monitor chain the *ending monitor*. We call the monitor marked with * the *ending positive monitor*, since it is the last monitor having positive nodes. When the ending monitor is not positive, we call it a *negative ending monitor*.

Each monitor maintains a “previously matched” table T_n for each node n assigned to it. The table T_n keeps the information on the combination of events and states that satisfy the timed conditions of and the temporal relationships among the nodes before n . In addition, each monitor maintains a history table H_v for each system view v in it. The history table H_v keeps the information on all of the events and states that have happened in the system view v .

The detection is event driven. For each monitor involved in a cooperative detection, three types of events can change its status: (1) history table update (HTU) event, (2) partially matched table update (PMTU) event, and (3) “false alarm update event”. The last event occurs only when the monitor is the ending positive monitor but not the ending monitor.

The HTU events refer to the raw events that occur on the target system. When an HTU event occurs, the monitor stores the event attributes and the timestamp into the corresponding history table. Then for each node n with which the system view is associated, the monitor tries the combination of the new event and “partially matched events” stored in T_n to determine whether they collectively satisfy the timed conditions of and the temporal relationships among the nodes up to n . When new combinations of events are found, the monitor raises a PMTU event for the next node in the chain. If the next node is positive and is in another monitor, the newly raised event is sent to the next monitor. A similar procedure will be executed when a PMTU event for node n occurs.

When the ending positive monitor finds a match on the last positive node, it raises an alarm both to the local console and the signature manager. When the negative ending monitor finds a match on the last negative node, it sends the corresponding information that identifies the combination of events to the positive ending monitor. The positive ending monitor then marks the corresponding alarm as false alarm.

Many optimization alternatives exist, but they are beyond the scope of this paper.

5. IMPLEMENTATION ISSUES

We are now implementing a CARDS prototype to verify our intrusion detection model and the designed architecture. Due to the strong capability of the Extensible Markup Language (XML), we have chosen this language to represent various specifications, such as system views and signatures. We have developed the corresponding Document Type Definition (DTD). A graphical user interface for designing signatures is also planned.

Various auditing mechanisms apply to specific security purposes, each of which may provide a different audit trail. For generality, our model is designed to be independent of any specific audit trail. The system views are the unique interfaces between the audit data and the detection engine. The probe is designed to restructure the raw audit data into the format of the system view; each probe is dedicated to a specific information source.

Probes are designed as modules that can be plugged into monitors. Whenever new information is needed, a new probe can be designed and implemented. The current implementation provides probes for Sun Solaris 2.x Basic Security Module (BSM) and traffic information captured by the TCPdump.

6. CONCLUSIONS AND FUTURE WORK

This paper describes the design of the CARDS for detecting coordinated attacks on large-scale, distributed systems. CARDS is a prototype for proving the signature-based detection model presented in [9], which represents coordinated attacks as generic event patterns over the structures of the typical information that can be found on target systems (i.e., system views). The system contains three types of components (signature managers, monitors, and directory services), which are distributed at various places in the network.

CARDS implementation is underway. We have chosen Java as the implementation language because of its rich API support. We have adopted Open LDAP to provide the directory service and XML to define a text-based signature specification language. As part of this effort, we would like to optimize the distributed detection algorithm by considering more application semantics.

References

- [1] J. S. Balasubramaniyan et al, An Architecture for Intrusion Detection using Autonomous Agents, TR 98/05, COAST Lab., Purdue, 1998
- [2] D. E. Denning, An Intrusion-Detection Model, Proc. 1986 IEEE Symposium on Security and Privacy, pages 118-131, Oakland, May 1986
- [3] A. K. Ghosh, J. Wanken, and F. Charron, Detecting Anomalous and Unknown Intrusions Against Programs, Proc. 14th Annual Computer Security Applications Conf., Pages: 259 -267 Scottsdale, AZ, Dec. 1998
- [4] T. L. Heberlein et al, A Network Security Monitor, IEEE Symposium on Security and Privacy, Oakland, CA, May 1990
- [5] H. S. Javitz and A. Valdez, The SRI IDES Statistical Anomaly Detector, IEEE Symposium on Security and Privacy, Oakland, CA, May 1991
- [6] J. Lin, X. S. Wang, and S. Jajodia, Abstraction-based misuse detection: High-level specifications and adaptable strategies. In Proceedings of the 11th Computer Security Foundation Workshop, pages 190-201, June 1998.
- [7] T. Lunt and R. Jagannathan, A Prototype Real-time Intrusion-detection System, IEEE Symposium on Security and Privacy, Oakland, CA, May 1988
- [8] B. Mukherjee, L. T. Heberlein, and K. N. Levvit, Network Intrusion Detection, IEEE Network, pages 26-41, May/June 1994
- [9] P. Ning, S. Jajodia, and X. S. Wang, A Scalable Signature-based Model for Detecting Coordinated Attacks. TR 01/00, George Mason Univ., 2000.
- [10] S. Northcutt, Network Intrusion Detection: An Analyst's Handbook, New Riders, 1999
- [11] P. A. Porras and P. G. Neumann, EMERALD: Event Monitoring Enabling Response to Anomalous Live Disturbances, Proc. 20th National Information Systems Security Conf., Baltimore, MD, Oct. 1997
- [12] S. E. Smaha, Haystack: An Intrusion Detection System, Proc. IEEE 4th Aerospace Computer Security Applications conference, Dec. 1988
- [13] S. R. Snapp, J. Brentano, et al, DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and An Early Prototype, Proc. 14th NCSC, pages 167-176, Washington, DC, Oct. 1991
- [14] S. Staniford-Chen et al, GrIDS - A Graph Based Intrusion detection System for Large Networks, Proc. 19th National Information Systems Security Conf. Vol.1, pages 361-370, Oct. 1996
- [15] G. Vigna and R. A. Kemmerer, NetSTAT: A Network-based Intrusion Detection Approach, Proc. 14th Annual Computer Security Applications Conf., Scottsdale, AZ, Dec. 1998