Proc. Research Conference, Troy, NY, October, 2002

Denial of Service Intrusion Detection Using Time Dependent Deterministic Finite Automata

Joel W. Branch
Dept. of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
brancj@cs.rpi.edu

Alan Bivens
Dept. of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
bivenj@cs.rpi.edu

Chi Yu Chan
Dept. of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
chanc4@cs.rpi.edu

Taek Kyeun Lee
Dept. of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
leet3@cs.rpi.edu

Boleslaw K. Szymanski
Dept. of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
szymansk@cs.rpi.edu

ABSTRACT

In this paper, we describe a new approach for the real-time detection of denial of service computer attacks using time-dependent deterministic finite automata. Current network-based intrusion detection systems employ state-transition based methods as a primary mean to detecting system penetrations and misuse as well. However, we utilize the time intervals between certain event occurrences [as defined in our automaton] to improve the accuracy of detecting specific denial of service attacks. Unlike some other detection systems, our design also lends itself to a distributed detection architecture, permitting non-obtrusive attack signature updating and operating system portability. This paper discusses the implementation of our prototype along with results from its test evaluation using publicly available data.

Keywords

Intrusion detection, computer security, denial of service attacks, computer networks, deterministic finite automata.

1. INTRODUCTION

Increasing attempts to compromise computer systems by methods ranging anywhere from masquerading as a privileged user to coordinating distributed attack probes across a network have led to increased research in intrusion detection. Traditionally, there have been two main classes of intrusion detection systems (IDS): host-based and network-based systems. A host-based IDS monitors the detailed activity of a particular host in real-time. The system call traces produced by an auditing mechanism such as the Solaris Basic Security Module (BSM) typically provides the IDS with the data needed to search for attack signatures [13]. When an analysis of the BSM data shows signs of an intrusion, the IDS alerts the system administrator of an attack. Proceeding host-based IDSes was the development of network-based IDSes. While some network-based IDSes focus on a single host, most typically monitor a network of computers and other devices (i.e. routers, gateways) that are subject to attacks. Subsequently, rather than using BSM data, network-based IDSes primarily use raw network packets to search for attack signatures throughout the network traffic. Our system is a network-based IDS.

Network-based IDSes, as well as host-based systems, can be further classified by different methods of protection: anomaly detection and penetration identification. The former method attempts to differentiate "anomalous" activity from the established normal operating behavior of a system. Our IDS falls under the last category: penetration identification (often referred to as misuse detection). After we define the "signature" of attacks in terms of sequences of system events and traffic data, we can identify these attacks in real-time and attempt to prevent any further damage to the system. Furthermore, by knowing precisely which attack we have detected, we can possibly tailor defensive strategies and/or system alarms appropriately.

In the approach described in this paper, we choose to focus on detecting one type of network attack: denial of service (DoS). DoS attacks are attempts to flood a network or an individual host with unwanted traffic in order to totally overwhelm the resources of the system. As a result, the system looses the ability to handle legitimate traffic/requests, experiences a loss in available bandwidth, and is overall unable to provide normal service to its users. In many cases, service is denied until the attacker's address is discovered so that further requests from that source can be blocked. However, often the source address is spoofed so that the true origin of an attack becomes untraceable. A more difficult scenario is the distributed denial of service (DDoS) attack, in which multiple clients are used to simultaneously flood a computer system with requests. In this case, more resources on the targeted system could be in jeopardy and more clients need to be blocked to recover from an attack. As with other attacks, it is most effective to detect DoS (and DDoS) attacks as early as possible.

In this paper, we will first explore some related IDS projects. Before entering a detailed discussion of the system's architecture, we will explain our specialized automata-based approach to penetration identification. We will then conclude with an evaluation of some system test results and also explain pursuable areas of future development.

2. RELATED WORKS

Due to the increasing complexity of computer networks as well as the growing sophistication of network-based attacks, network-based intrusion detection has recently gained more attention from academic, military, and commercial sectors. Consequently, various IDSes have been implemented that address different aspects of network security and use different methods of detection. For example, some tools employ artificial neural networks, trained for misuse detection across a network [2]. Another system, GrIDS, constructs activity graphs from network traffic data to detect large-scale automated attacks in real-time [11]. EMERALD uses a scalable distribution of surveillance monitors throughout a network to apply distributed event correlation models in detecting network intrusions [8].

Our system shares methodologies similar to those of a couple of other IDSes. For instance, Snort is a lightweight network-based IDS that utilizes a rules-based approach, along with network packet-sniffing and logging to perform content pattern matching and detect a variety of attacks [9]. In turn, we also use a rules-based approach in detecting attacks, but we apply it in the form of a deterministic finite automaton, which considers the actual sequence of rules needed to define a probable attack. Another comparable tool is NetSTAT. NetSTAT is an implementation of the STAT design which models an attack as a sequence of actions which progressively takes a computer from an initial normal state to a compromised state [4, 14]. NetSTAT applies that model to a networked environment by modeling both the guarded network *and* the attacks. In doing this, it determines which network events have to be monitored and where in the network they should be monitored. However, we bypass modeling the network and focus directly on the representation of various DoS attacks. We also consider the time intervals between system events when defining the transitions between a computer's states.

3. TIME- DEPENDENT DETERMINISTIC FINITE AUTOMATA APPROACH

3.1 Time-Dependent Deterministic Finite Automata

Conceptually, a deterministic finite automaton (DFA) is a computational model designed to represent an idealized computer. Just as a computer changes states of operation and produces some outputs given particular inputs, so does a DFA. Specifically, DFAs are designed to recognize member strings of a specified language. More formally, they recognize those languages belonging to the class of *regular* languages [10]. In our particular case, we want to recognize the language of DoS attacks (to be explained in section 4.2). DFAs carry two important properties. First, they embody a *finite* number of states. Second, they are deterministic, meaning that given a current state and an input, the automaton "transitions" to only one state (which could be the same state or a new one).

DFAs are usually represented by state-transition diagrams. Circles represent the automaton's different states and unidirectional arrows represent inputs that may allow transitions between different states. Typically, final states, in which the entirety of an input string is accepted, are double circled. Also, for every state, there must be an exiting transition for every character defined in the language's alphabet. Figure 1 is an illustration of a simple DFA which accepts inputs of the alphabet {a, b} containing the string "aba". In this example, q1 is the start state and q4 is the final state. If the DFA is in state q1 and receives "a" as an input, then the DFA travels to state q2. If while in state q2 the DFA receives "b" as an input, then it travels to state q3. If an "a" is received while in q3, the DFA then travels to the final state and the input is accepted. An examination of figure 1 shows that any input disrupting the pattern "aba" sends the DFA back to its start state. While the DFA in figure 1 has only one final state, this is not a specification for DFAs in general. A DFA can have more than one final state.

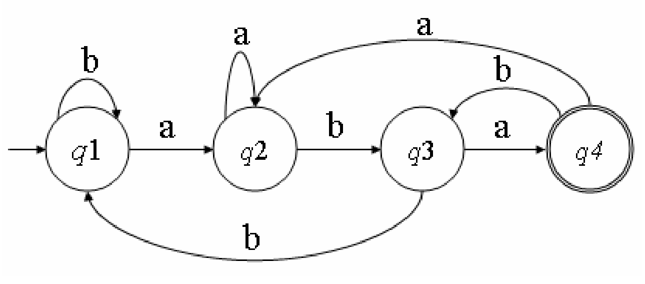

Figure 1. An example DFA.

A time-dependent deterministic finite automaton (TDFA) is very similar to the machine we just described. However, it considers more than just the sequence of inputs; a TDFA also considers the time intervals between inputs in recognizing members of a language. This becomes very beneficial in the use of automata to recognize DoS attack signatures since many DoS attacks are dependent upon the time intervals between arriving network packets. Figure 2 shows an example TDFA. We can think of this machine as recognizing the pattern "a, b<5, a<5." In other words, the "b" must occur within five seconds of the initial "a", and the last "a" must occur within five seconds of the "b". All transitions shown without the five second time restraint are *default* transitions. If the desired input does not occur within the specified time restraints, then the default transition sends the TDFA back to the appropriate state where it can continue monitoring for the required pattern. The same occurs for input symbols that do not match the required character (i.e., receiving a "b" when either "a<5" or "a" is expected). Notice that the transition from q1 to q2 is a default transition even though it takes the TDFA closer to the final state. This is because this "a" is the *first* character in the pattern; it does not make sense for it to come within five seconds of some other input.

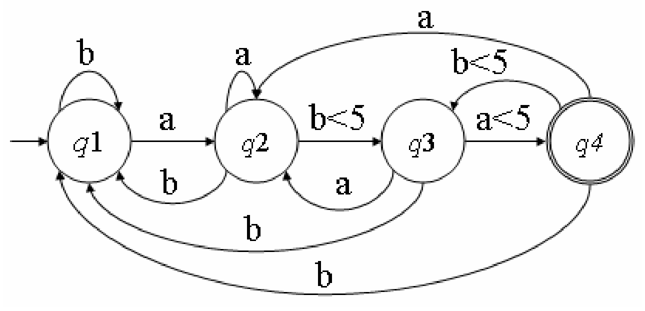

Figure 2. An example TDFA.

3.2 DoS Attack Representation Using Time-Dependent Deterministic Finite Automata

The very nature of TDFA models makes it a logical choice in representing DoS attacks. DoS attacks can aptly be viewed as a characteristic series of network events or special packets that render a particular resource inoperable. Therefore, we use the transitional arcs of TDFAs to represent those characteristic attack events (or packets). TDFA states are used to represent incremental conditions of a system as it reaches a state of intrusion. The final state(s) of a TDFA then represent points of attack completion.

4. SYSTEM ARCHITECTURE

Now, we will discuss the full architecture of our system, including its prime functions and its various subsystems and their interactions. Before continuing, it would be useful to note that our system detects DoS attacks occurring in TCP, UDP and ICMP network traffic. These are all industry-standard network communication protocols [12]. The following are the key characteristics of our IDS:

- Our system detects DoS attacks from both real-time and historical data
- It uses time-dependent deterministic finite automata to model and confirm attacks
- It supports the updating of attack models without interruption to other system components

Having the option between operating from real-time or historical data offers a couple of advantages for a site security officer (SSO). Real-time monitoring of network traffic provides the best level of protection because *ongoing* attacks may be prevented. However, with the addition of pre-recorded (historical) datasets, offline operation still allows the SSO to see if and when attacks might have occurred while the IDS was down for maintenance. Also, the SSO can use experimental datasets to test and tune the system for newer attacks. Next, while confirmation procedures will be discussed in greater detail later, it is important to further note the significance of time-dependent deterministic finite automata in our system here. Earlier, we discussed how TDFAs serve as an appropriate model for DoS attacks. However, the additional benefit of TDFAs for site security officers is that they permit the storage of both an attack's base signature and its variation(s) using only one model construct. A SSO can even use one TDFA model to represent multiple DoS attacks, as we do in our implementation. Additionally, using time-based information in attack signatures increases the accuracy of detecting DoS attacks. Last, the ability to update attack models without disturbing other system components prevents degradation to the system's performance during detection. As will be discussed later, this feature also presents the opportunity for an agent-based distributed architecture. Also worthy of noting is our selected development language, Java v1.2. Due to Java's platform independence, our program can easily be run on any operating system with a Java virtual machine installed.

At present, four distinct components make up our system: (1) the data filtration unit, (2) the event token generator, (3) the TDFA transversal unit, and (4) the TDFA provider. External components with which our system interacts are a local area network (LAN), stored (historical) network traffic data, a client machine. The connectivity of the entire system is illustrated in figure 3.

4.1 Data Filtration Unit

Network packets carry a wealth of information (i.e. sequence number, header length, checksum, etc.), all of which are not needed for the purposes of all network-based IDSes. Since our focus is on DoS attacks, the packet data

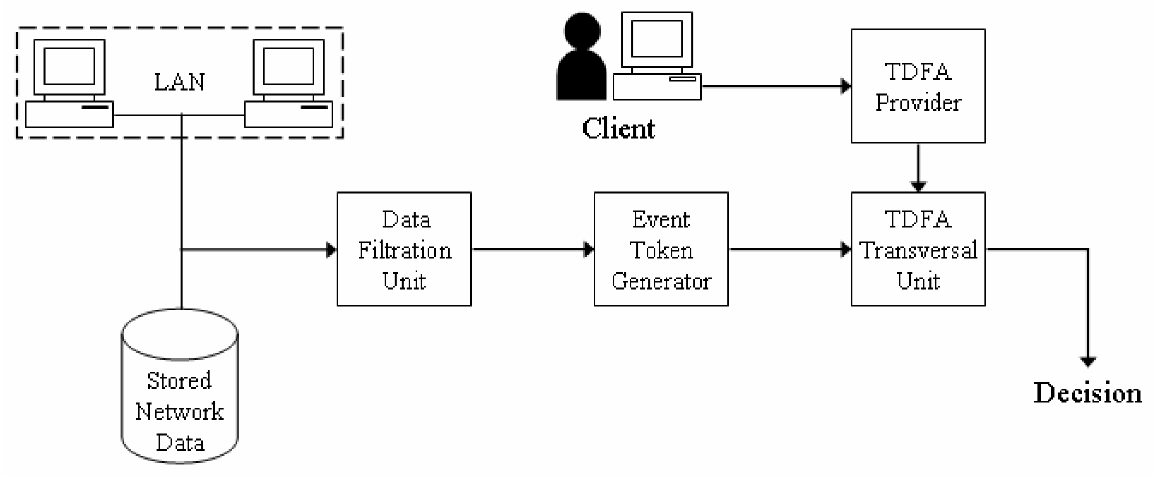

Figure 3. Overview of system architecture.

fields we are interested in are those pe rtaining to such things as source and destination addresses and also the various flag fields (i.e., SYN and ACK). The function of the data filtration unit (DFU) is to process relevant network packet information for subsequent components of the system (according to the flow of data).

As seen in figure 3, network traffic data originates from either of two sources: a local area network (LAN) or a stored data source. We mentioned before how the Solaris BSM utility supplies audit data for host-based IDSes. However, the utility module we use is tcpdmp [5], which can provide a record of network activity for a particular machine in ASCII text form, delimited into various fields. The following are fields for which the DFU parses:

- Packet type
- Source IP address
- Destination IP address
- Destination port
- More fragments flag
- Timestamp
- SYN flag
- ACK flag
- Echo request
- Echo reply

Live data originating from connected LAN devices are used in real-time detection while stored data, our second source of extraction, is used in offline mode. As opposed to the prior source, stored data usually resides in a log file and may or may not be in ASCII, but binary (non-text) form. Since the data filtration unit is designed to process packet information only in ASCII form, any binary information must first be converted to the proper format before it can be processed. The end product is a delimited ASCII text message which contains specific network event information (including TCP, UDP and ICMP packet data).

4.2 Event Token Generator

After network event data is processed by the DFU, the corresponding ASCII text (remaining in a delimited format) is sent to the event token generator (ETG). The ETG is then responsible for translating the DFU text messages, each representing a particular network event, into special tokens. We must note that there is not necessarily a "one-to-one" relationship between DFU messages and ETG tokens. It is quite possible that the information in one message will cause the ETG to generate a "sequence" of tokens. Together, all of these predefined tokens, each being a string of one or more ASCII characters, compose a language used by our system for recognizing DoS attacks. The efficiency of domain-independent IDS languages has been shown with such languages as STATL [3] (used by both USTAT and NETSTAT), which partially serves as our motivation for using a proprietary language. Our other incentive is that it drives the operation of our detection engine: the TDFA transversal unit (to be explained later).

The following is a condensed explanation of how the ETG operates. Suppose that after reading the DFU text message, the ETG determines that the UDP destination port number of the packet is set to an echo port number (a condition indicative of a UDP Storm attack). Subsequently, the ETG generates the token "e" and sends it to the TDFA transversal unit, where it will be used for [UDP Storm] attack recognition. Table 1 highlights just a few of the tokens that compose the language for our system.

4.3 TDFA Transversal Unit

Most intrusion detection systems have some distinguishable core component primarily responsible for recognizing

Table 1. Sample ETG tokens and their definitions.

Token(s)	Definition
S	Packet's SYN flag is checked
F	Packet's MF flag is checked
J>5	Non-initial ACK packet; time interval between this and current packet is greater than 5 seconds
&	First ICMP echo reply packet to a particular destination [address and port]

Table 2. DoS attacks sought for by our IDS.

Name of attack	Protocol used	Effect
Land	TCP	Operating system loops and eventually freezes
SYN Flood	TCP	Legitimate service requests are denied as CPU resources become totally consumed; operating system may crash or loop
Ping Flood	ICMP	Network slows down; network connectivity may be disabled
Process Table	TCP	Process table is completely filled with network server instantiations; new processes cannot be started
Smurf	ICMP	Host floods both itself and intermediate network with ICMP echo replies
Teardrop	N/A	Host may hang or crash
UDP Storm	UDP	Legitimate service requests are denied as CPU resources become totally consumed; network may become congested

attacks. In our system, the TDFA transversal unit (TTU) acts as th e main attack detection engine. The TTU is what actually embodies the TDFA that represents the various DoS attacks to be detected.

The relationship between the TTU and the previous module, the event token generator, is best thought of as that between a physician and patient. The EVT (patient) displays "symptoms" [on behalf of the guarded host] of probable DoS attacks. As discussed earlier, these symptoms materialize themselves as tokens. The intent of the TTU (physician) is to read the tokens and "diagnose" the host as being under, or not under a state of attack. It does this by using the EVT tokens as input characters to traverse the supplied TDFA. When the TTU finds that its TDFA has reached a final state, it will alert the site security officer that an attack (specified by its respective final state) has occurred.

As we mentioned in section 3, it would be beneficial for a SSO to verify the effectiveness of the IDS using pre-recorded network traffic data containing traces of successful intrusions. Specifically, this entails verifying the correctness of the TDFA transversal unit. Table 2 lists seven DoS attacks we designed our TDFA to recognize.

4.4 TDFA Provider

We mentioned earlier in section 4 how our system supports the updating of TDFA attack models without interruption to the rest of the system. The TDFA provider is what makes this feature possible. When a site security officer (client) wants to replace the resident TDFA, it interacts with the TDFA provider in giving it a description of the new TDFA model. However, models such as those depicted in figures 1 and 2 are not required. A client has only to specify the attack "signature", meaning only those states and transitions that lead directly to attack completion. All other transitions, such as those sending a TDFA back to its start state, would automatically be supplied by the TDFA provider. This is convenient for the client as he or she needs only to provide a simple, linear attack model. The TDFA provider then supplies the TDFA traversal unit with the user-defined TDFA description. The passing of a new attack model can occur without degradation to other components of the system because the TDFA provider interacts only with the TDFA transversal unit. It should be noted that at start-up time, the TDFA transversal unit does contain a default TDFA model, so detection is possible before interjection by the SSO. As will be described with more detail in section 6, this component allows the opportunity to develop a distributed modular architecture permitting a more automated approach to updating attack signatures.

5. TEST RESULTS

In keeping with the standard measure of most other intrusion detection systems, we used the datasets from the Defense Advanced Research Projects Agency (DARPA) Intrusion Detection Evaluation in testing our IDS. Specifically, we used the publicly available datasets from both the 1998 and 1999 evaluations [7].

In total, we used 8 tcpdump training data files: 5 from 1998 and 3 from 1999. Currently, we have tested our system on 5 of the 7 attacks we listed earlier. The results of the test are highlighted in table 3. The third column shows the timestamp of the packet finalizing the specified DoS attack, according to the DARPA dataset. The fourth column shows when our system recognized the specified attack. As is visible from the table, we were very accurate in detecting five various DoS attacks. However, in one dataset, 1998_week6_fri, we did fail to detect the SYN Flood attack. The attack was successfully detected in most other datasets, although there was also a problem with the 1998_week4_tues dataset. We falsely detected the SYN

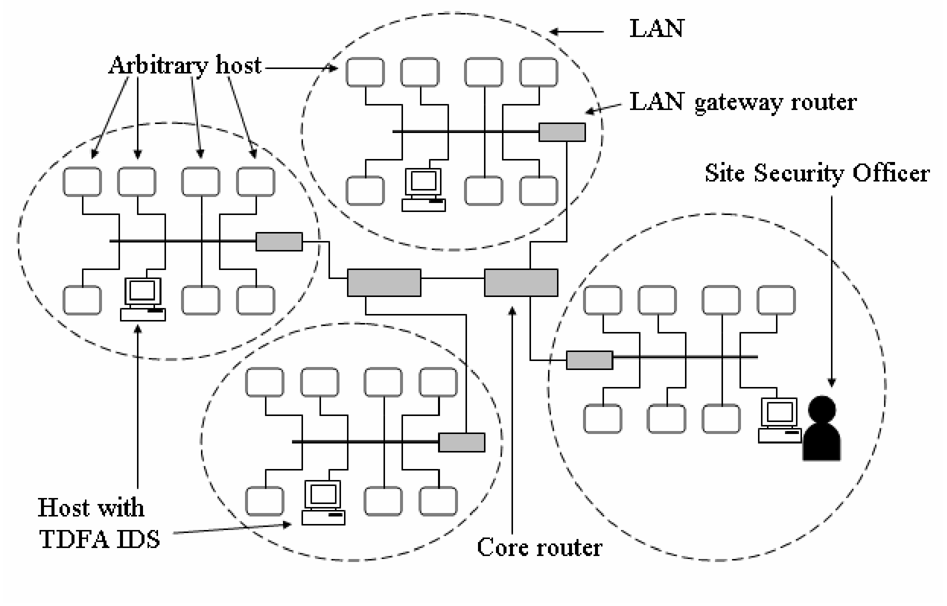

Figure 4. Overview of proposed modular architecture.

Table 3. Test Results.

Dataset	Attack	MIT Time	TDFA Time
1998_week4_tues	SYN Flood	11:55:38	8:50:15
	Ping Flood	20:11:31	20:11:31
	Teardrop	23:15:08	23:15:08
1998_week5_mon	Teardrop	08:15:02	8:15:02
	Smurf	12:53:15	12:53:15
	Smurf	15:33:28	15:33:28
1998_week5_fri	SYN Flood	17:27:07	17:27:07
	Smurf	18:00:15	18:00:17
1998_week6_tues	Ping Flood	13:04:56	13:04:56
	Land	17:53:49	17:53:49
1998_week6_fri	Teardrop	08:32:12	8:32:12
	SYN Flood	09:31:52	NO
	Smurf	19:12:37	19:16:27
1999_week2_mon	Ping Flood	08:50:15	8:50:15
	Land	15:57:15	15:57:15
1999_week2_thur	SYN Flood	11:04:16	11:04:16
	Land	15:47:15	15:47:15
1999_week2_fri	Ping Flood	09:18:15	9:18:15
	SYN Flood	11:20:15	11:20:15

Flood attack approximately three hours before it supposedly occurred—a “false positive”. In turn, this also presents a “false-negative” because we still failed to detect the attack at the correct time (just as with the 1998_week6_fri dataset).

6. CONCLUSION

We were pleased with the results from our system evaluation. However, we are concerned about the lack of SYN flood attack detection in two of the 1998 datasets. We are considering the fact that the TDFA we used in modeling the attacks did not embody most variations of the attack signatures. Currently, we are studying the DARPA datasets in attempts to improve our TDFA model. We are also in the process of testing our system for the remaining attacks: process table and UDP storm.

We also acknowledge that our IDS, being a misuse detection system, has a significant flaw: it can only detect attacks for which it knows a signature. Detecting malicious activity by way of signature pattern, however, has been a widely supported practice, such as in anti-virus utilities. Furthermore, it decreases the number of false-positives in the detection process, which is very important for performance.

In general, network-based intrusion detection is still a relatively young field of study in computer science. As networks increasingly become more complex, the need for sophisticated security tools will rapidly grow in importance. While our IDS does not detect all malicious network penetrations, it does a great job detecting a significant subset of these attacks: denial of service. Evidently, from the test results, considering the time intervals between network events (in DoS attacks) is worthy of pursuit. We contribute to the network security field by offering a detection tool that is based off of the temporal characteristics of these attacks. Our IDS detects these attacks in an accurate and efficient manner and is compact enough to be coupled with other IDSes (perhaps even anomaly detection systems) to build a complete suite of general attack detection/prevention tools on multiple platforms.

7. FUTURE WORKS

We mentioned earlier the opportunities that the TDFA provider component can offer us in the area of distributed attack detection. Currently, this serves as the focus of our next area of further development pertaining to this system.

Our application, as well as other intrusion detection systems, uses "sniffed" network traffic to determine significant intrusion events. However, sniffed data traditionally can only be obtained from a packet sniffer running on the same network as the monitored host machines. When the monitored hosts are in different networks, multiple sniffers are needed. The growing degree of segmentation in networks has caused problems in many intrusion detection applications relying on network sniffed data [6]. To facilitate distributed detection, it is important to recall the modular framework of our application such as that portrayed in figure 3. Modularity such as this would be necessary to distribute the functionality of our system. Our intent is to use the DOORS system (Distributed Online Object Repositories) in which our detection process would be encoded into mobile agents and sent to the network in need of monitoring [1]. Figure 4 illustrates the topology of such a network scheme. This figure is a portrayal of a large network, similar to the Internet, in which smaller LANs are connected by gateway and core routers. In this case, a host in each monitored network will be equipped to receive an agent containing our TDFA IDS. A site security officer will update the host running the IDS with a new TDFA structure when necessary by sending a new TDFA to the TDFA provider module shown in figure 3. This module would then immediately replace the TDFA structure used by the TDFA transversal unit to determine attacks. We believe this would effectively distribute the detection while providing a centralized management for continual TDFA updates.

REFERENCES

[1] J. Bivens, P. Fry, L. Gao, M. F. Hulber and B. Szymanski. Agent-based Network Monitoring. Agent Based High Performance Computing Workshop, Agents '99 Conference. Seattle, Washington, May 1999.

[2] J. Cannady. Artificial Neural Networks for Misuse Detection. In *Proceedings of the 1998 National Information Systems Security Conference* (NISSC'98), pages 443 - 456, Arlington, VA, October 1998.

[3] S. T. Eckmann, G. Vigna, and R. A. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. Dept. of Computer Science, University of California, Santa Barbara, 2000.

[4] K. Ilgun, R. Kemmerer, and P. Porras. State Transition Analysis: A Rule-Based Intrusion Detection System. IEEE Transactions on Software Engineering, 21(3), March 1995.

[5] V. Jacobson, C. Leres, and S. McCanne. *tcpdump*, June 1989. Available via anonymous FTP from *ftp.ee.lbl.gov*.

[6] S. McCloghrie and J. Scambray. Once-Promising Intrusion Detection System Stumbles Over Switched Networks. *InfoWorld*, volume 22, page 58, InfoWorld Media Group, Inc., December 2000.

[7] MIT Lincoln Laboratory. DARPA Intrusion Detection Evaluation. *http://www.ll.mit.edu/IST/ideval/*, 1999.

[8] P.A. Porras and P.G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353 – 365, Baltimore, Maryland, October 1997.

[9] M. Roesch. Snort – Lightweight Intrusion Detection for Networks. Snort – The Open Source Network IDS. *http://www.snort.org/docs/lisapaper.txt*.

[10] M. Sipser. Introduction to the Theory of Computation. PWS Publishing Company, 1997.

[11] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS - A Graph Based Intrusion Detection System for Large Networks. Proceedings of the 19th National Information Systems Security Conference, volume 1, pages 361 - 370, October 1996.

[12] W. Stevens. UNIX Network Programming, Volume 1, Second Edition. Prentice Hall PTR, 1998.

[13] Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303, USA. *SunSHIELD Basic Security Module Guide, Solaris 7*, October 1998. Part No. 805-2635-10.

[14] G. Vigna and R. A. Kemmerer. NetSTAT: A Network-based Intrusion Detection Approach. In *Proceedings of the 14th Annual Computer Security Conference*, Scottsdale, Arizona, December 1998.