IntruDetector: A Software Platform for Testing Network Intrusion Detection Algorithms

Tao Wan Xue Dong Yang
Email: {taowan, yang}@cs.uregina.ca
Department of Computer Science
University of Regina
Regina, Saskatchewan S4S 0A2
CANADA

Abstract

An Intrusion Detection System (IDS), that monitors passively specific computing resources, and reports anomalous or intrusive activities, is becoming an important component in the security system of information infrastructure. Algorithms for detecting intrusions are under rapid development, but far from being mature. One interesting and difficult issue is how to study and test a new intrusion detection algorithm against a variety of (perhaps simulated) intrusive activities under realistic background traffic. A flexible and general-purpose platform for testing intrusion detection algorithms is clearly desirable. This paper presents such a software platform, called IntruDetector. With this platform, detection algorithms can be tested directly in a real environment with wide range of intrusive activities. The data of normal system activities are directly collected from the live environment, and are mixed with intrusive activities that are simulated by hybrid simulation. The main properties of this approach are: (1) the background traffic is realistic; (2) it allows flexible simulation of various types of intrusions; and (3) normal system operation will not be disrupted by virtually simulated destructive intrusions during testing.

1 Introduction

As the usage of the Internet is proliferating, more and more of our daily lives and business operations depend on the availability of networks and the integrity of data. Unfortunately, the information infrastructure on which we depend is very vulnerable to attacks. Computer intrusions are occurring from day to day, and have become a major threat to our society [1, 2, 3]. How to detect intrusions is a big challenge faced by every organization.

An Intrusion Detection System (IDS) is a system that monitors passively specific computing resources, such as operating systems, applications, or networks, and reports any anomalous events or any known patterns indicating potential intrusions. IDSs have been receiving more and more attention from both research institutes and industrial sectors. Although certain progress has been made in this area, it is still a comparatively young and far from being mature. Many technologies from other fields are being applied and various detection systems are under development [5, 7, 11, 12, 14, 15, 16, 17]. Naturally, researchers strongly need an environment for testing and evaluating their detection systems.

Researchers at University of California at Davis, IBM, and MIT Lincoln Labs have made significant efforts on testing and evaluating IDSs. Their works are briefly summarized as follows.

(1) Testing at UC Davis

The researchers at the University of California at Davis have been the first in literature to address the issues of testing intrusion detection systems [22, 23]. They have developed a methodology and software platform for testing IDSs. UNIX package, Expect, was employed to develop the scripts for simulating intruders as well as normal users. Intruders can be individual or cooperative. Three testing objectives are identified as follows:

- Broad Detection Range: the ability to distinguish intrusions from normal activities.
- Economy in Resource Usage: the efficiency of using system resources.
- Resilience to Stress: the ability to function correctly under high load.

Network Security Monitor (NSM) [13, 21] was evaluated in the environment as an example.

(2) Testing at MIT Lincoln Labs

MIT Lincoln Labs conducted two IDS tests in 1998 and 1999 respectively [4, 19]. The tests were performed in a simulated network. The test data were sampled from a real environment. The pseudonymized data were replayed in the simulated network by scripts or manually duplicated transactions. Several well-known attacks were simulated and inserted into background traffic. Several major IDSs were tested, and the test results were also published.

This test is sophisticated and advanced, and the results can be used as a basis to compare different IDSs. The test data are comprehensive and are being used by many researchers for experimenting their IDSs.

(3) Testing at IBM Zurich Research Lab

IBM Zurich Research Laboratory has developed an experimentation workbench for testing intrusion detection systems [8]. An experimentation network was constructed, which was disconnected from the rest of the world. Both normal user behaviors and intrusions are obtained by simulation. Normal user behaviors are simulated by scripts written in Expect. Intrusions are simulated by the exploit scripts from a vulnerability database maintained by the lab internally. After examining the analysis results of various IDSs, the operator can therefore evaluate their performance.

All these efforts focus on testing the performance of an IDS consisting of several discrete components, for example, sensors, analyzers, among others. Their results can be used by end users as guildlines to select IDSs, or by vendors to improve their products.

Our work is enlightened by the above efforts but different from them. Instead of focusing on testing an IDS, we try to build a software platform for testing intrusion detection algorithms incorporated by an IDS. We first describe the components of which an IDS may consist, then give the rational why a platform for testing detection algorithms is of interest.

According to the Intrusion Detection Exchange Format (IDWG) [10], an under developing standard for IDS, an IDS consists of a set of discrete components: **sensors, analyzers, managers, among others**. Sensors process data sources (for example, raw network packets, operating system audit logs, application logs, etc.), and generate events representing occurrences of activities. Normally, sensors collect data and reduce them to a certain degree since the volume of the data are normally tremendous. Analyzers analyze events produced by the sensors and report security alerts for unauthorized or undesired activity. Analyzers utilize intrusion detection algorithms to perform analysis. For example, analyzers of a misuse IDS may implement a pattern-matching algorithm, but analyzers of an anomaly IDS may utilize neural networks to build profiles. In this paper, intrusion detection algorithms and analyzers are used interchangeably. Managers can be used to perform configuration, presentation, or reporting.

We see a platform for experimenting intrusion detection algorithms is of interest in that:

(1) Among the components of which an IDS may consist, analyzer is deemed to be very important since the efficiency of an IDS is mainly determined by how efficient its detection algorithm is. Therefore, a detection algorithm needs extensive tests before being integrated into an IDS.

(2) Testing an IDS component by component conforms to the principal of modularity in software engineering. It makes it easy to debug and maintain an IDS. Although many existing IDSs mix together the functionalities of components like sensors and analyzers, we think a clear separation of functionalities among IDS components is beneficial.

(3) Researchers who are interested in developing intrusion detection algorithms need a flexible platform to evaluate their algorithms. Currently, many researchers are using the data set provided by MIT's Lincoln Labs to test their algorithms. However, some researchers [12] have reported that the intrusive data contained in that data set are limited, and there is no way for them to insert new intrusive data. We try to provide an alternative tool for flexible simulation of intrusions.

(4) To detect distributed intrusions (for example, Distributed Denial of Services), attack correlation is of extreme importance. A platform allowing for flexible simulation of distributed attacks shall be interesting to the developers of intrusion correlation algorithms.

A testbed for intrusion detection algorithms should fulfill but not limit to the following requirements: (1) It should produce sufficient and realistic background data; (2) It allows for flexible simulation of a variety of different intrusive activities; (3) Testing environment should be free from any potential dangers caused by simulated attacks; (4) The tested algo-

rithms should be fed with the same data set so that their analysis results are comparable. The limitations of current methodologies are: (1) Background traffic is either purely simulated or recorded from a real world and replayed in the experimental environment. As a result, it is not realistic or sufficient, or the overload of recording and replaying is heavy. (2) Destructive intrusions can not be emulated since they may compromise the experimental environment.

To address the limitations as stated above, we designed and implemented a software platform, called IntruDetector, for testing network intrusion detection algorithms by applying the technique of hybrid simulation. IntruDetector allows for test to be conducted directly in a real environment without compromising normal system operations. Background data representing normal system activities are collected directly from a real environment, and intrusive data are generated by hybrid simulation. Destructive intrusions are virtually simulated and non-destructive intrusions can be generated by real simulation. Intrusive data, mixed with normal data, are sent to a central repository where they will be retrieved and analyzed by intrusion detection algorithms. All algorithms under test are ensured to have a consistent view of data, so their analysis results are comparable on the same basis.

The rest of the paper is organized in the following manner. In Section 2, the testing environment is discussed and the technique of hybrid simulation is explained. The architecture and implementation of IntruDetector are presented in Section 3. In Section 4, we describe some preliminary results of the experiments we have conducted with IntruDetector. Topics for future research are outlined in the last section.

2 Testing Environment

Testing of intrusion detection algorithms can be performed either in a real environment or an experimental environment. The advantages and disadvantages of each method are discussed below and our choice is then presented.

2.1 Testing in a Real Environment

Testing IDSs can be conducted in a live environment where many real users produce significant background traffic by using a variety of network services, e.g., mail, ftp, telnet, etc. In this method, background traffic is directly collected from the real environment, and intrusive activities are emulated by exploit scripts. The advantage of the method is that background traffic is sufficiently realistic. It also eliminates the need to analyze the pattern of normal user activities and the effort to develop corresponding simulation tools. Therefore, the workload of tests can be significantly reduced. However, the method has the following drawbacks:

- The testing environment is exposed to the risk of attacks from both inside and outside. Since the test is performed in a real environment which is normally connected to the Internet and is technically accessible from anywhere at anytime, it could be compromised by attacks from both inside and outside.

- It is possible that background traffic may contain intrusive data as attacks against the testing environment could occur during the time frame of testing. This possibility could increase the inaccuracy of test results. For example, if the attacks similar to those contained in the background traffic are emulated to test an anomaly detection algorithm, they may not be flagged as abnormal since they are part of the training data. To experiment misuse detection algorithms with the background traffic containing intrusive data, we may deem the reports of these attacks as false alarms since their occurrence is unknown to us.

- Normal system operation could be interrupted by simulated attacks. To test an algorithm, we need to emulate as many intrusions as possible to test its ability to identify them from background traffic. When destructive attacks are emulated, the normal system operation could be compromised.

2.2 Testing in an Experimental Environment

Due to the high risk to perform testing in a real environment, most researchers perform their tests in experimental environments [8, 19, 22]. The challenge of this method is how to produce realistic background traffic in an experimental environment. Three ways of generating background traffic are summarized here. First, the background traffic can be manually produced by employing persons to perform whatever occurred in a real environment. Second, it can be obtained by simulation scripts which generate data conforming to the statistical distribution of background traffic occurred in a real environment. Third, background traffic can be dumped from a real environment and replicated

in the experimental environment. In order to emulate intrusions, software packages can be used to develop scripts to explore system vulnerabilities. Exploit scripts are then run in the experimental environment to emulate intrusive activities.

The disadvantages of this method are :

- The overload to manually produce background traffic in an experimental environment is heavy as there are normally hundreds or thousands of users in a real environment.
- The behavior of real users are difficult to model, and the statistical distribution of background traffic is not agreed. Some researchers claimed that it conforms to Gaussian distribution, but some others found it is not [13].

2.3 Our Approach

We performed testing in a real environment based on the following reasons.

(1) Background traffic must be sufficiently realistic. One of the important testing objectives is to measure the ability of detection algorithms to identify intrusions from normal activities. Without realistic background traffic, test results would be inaccurate. However, it is difficult to generate realistic background traffic in an experimental environment as stated above.

(2) If the testing environment is configured very carefully, the risk of exposing it to attacks from inside and outside is low, and the inaccuracy of testing results brought by the possibility that background traffic may contain intrusive activities is not significant.

The main challenge of conducting testing in a real environment is how to protect normal system operations from being compromised when attacks are simulated. This is where the idea of hybrid simulation fits.

Computer simulation plays a fundamental role in many research fields. It is utilized for a variety of different reasons, mainly because either the cost or risk is prohibitively high, or it is completely impossible to do an experiment in a real environment. Although simulation has been widely applied to many areas of computer science, such as network protocol design, routing algorithm development, performance evaluation, among others, limited work has been done in the field of testing intrusion detection systems.

In order to apply the technique of simulation, we need to choose from exact simulation or abstract simulation. There are always trade-offs between these two approaches. Exact simulation can produce detailed and realistic results, but abstract simulation can simplify the process and concentrate on the certain level of interest. We would like to take the advantages of both approaches, and apply them to different scenarios accordingly. To simulate attacks which do not interrupt the activity of normal users or affect the operation of testing environment, exact simulation is utilized. We can develop exploit scripts by software tools, such as UNIX shell, Expect, Tcl, etc. We can also borrow intrusive tools used by attackers. Simulated intrusive traffic along with the background traffic will be captured by a sensor. The sensor then generates high level synthesized events and passes them to detection algorithms for further analysis. Exact simulation is also called real simulation here since it generates real network traffic. To simulate intrusions which are destructive to the testing environment, abstract simulation is used. Abstract simulation is also called virtual simulation since it emulates intrusions in a high level and does not produce any real network traffic. To virtually simulate an intrusion, we need to study the system vulnerability it explores and the pattern of traffic it generates. From there, a simulator which can generate a high level of events corresponding to the intrusive activity shall be built. The outputs of virtual simulators and the outputs of the sensor are mixed together in a central place where detection algorithms will retrieve them and perform analysis. This combination approach is called hybrid simulation.

We use SYN-Flooding [24] denial of service attack as an example to illustrate the operation of hybrid simulation. We also show that virtual simulation is functionally equivalent to real simulation when being used to test detection algorithms, but with the advantage of protecting normal system operations from being compromised.

SYN-Flooding is a denial of service attack. It explores a weakness of TCP/IP. In TCP/IP, a three-way handshake protocol is used for establishing a connection. The initiator sends out a SYN segment to the receiver to build a connection. After receiving the SYN packet, the receiver will send back a SYN ACK segment. At the same time, some memory space is allocated to store the status information of the connection to be completed. Normally, the initiator will send back another ACK segment to the receiver, and the connection is therefore successfully established.

To launch a SYN-Flooding attack against a machine, the attacker transmits a huge number of SYN packets to the victim machine in a very rapid rate,

but does not send back any ACK segments. In this case, the victim has to keep the status information of uncompleted connections for a while. If the number of uncompleted connections exceeds the system limit, the victim may run out of its memory, and will not be able to respond to legitimate requests. In the worst scenario, it must be shutdown.

1. Real Simulation

To apply real simulation to emulating this type of attack, we can write a program to open a raw socket, compose SYN packets, and keep sending them out to a target quickly. To spoof the target, the sources of SYN packets can be set to any arbitrary addresses. In this way, the SYN ACK segments sent back by the target will be lost and the target will never be able to receive any acknowledgments. The target has to keep status information for uncompleted connections in its memory until a timer expires. If the number of uncompleted connections exceeds its limit, the target will not be denied of services in terms that it is unable to serve any legitimate requests.

The SYN packets generated by the real simulator will be captured by the sensor. The sensor then generates a series of synthesized events called TCP connection establishment request events. These events, mixed in the events representing background traffic, are sent to a central repository called the event buffer.

2. Virtual Simulation

To virtually emulate the SYN-Flooding denial of service attack, a simulator is developed to directly generate a series of high level TCP connection establishment request events and send them to the event buffer. In the event buffer, virtually simulated events are injected into background events produced by the sensor.

Since the event buffer is the only place where intrusion detection algorithms read events, it is easy to see that virtual simulation is functionally equivalent to real simulation in terms that they can produce equivalent inputs to detection algorithms.

The advantages of our approach are:

- Background data are realistic and sufficient since testing is conducted directly in a real environment.
- A broad range of intrusions can be emulated, including destructive and non-destructive intrusions.
- Normal system operation is protected since destructive intrusions can be virtually emulated.

3 The Software Platform

IntruDetector consists of several components: a sensor, simulators (real and virtual), an event buffer, and a visualization engine. The system architecture is depicted by Figure 1.

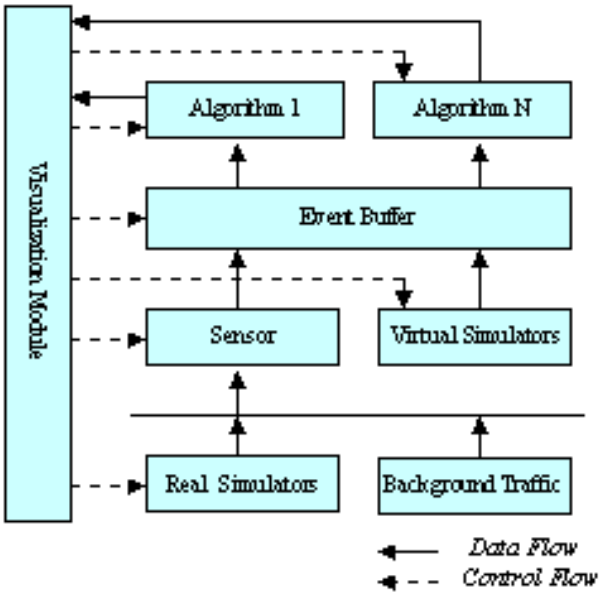

Figure 1: The System Architecture of IntruDetector

The sensor should implement the functionalities described by IDWG. It processes raw data and produces events in the format conforming to current standards, for example, Intrusion Detection Message Exchange Format (IDMEF) [10]. The sensor should also reduce data to a certain degree. We have implemented a sensor for processing network traffic. The sensor captures network packets and generates network events based on the headers of packets. Packets are captured by calling the libraries provided by libpcap [20]. The current format of events generated by the sensor does not conform to any standards since they are still under-development. We will keep our eyes on the progress of IDS standardization, and redesign our platform when they are finalized.

It should be pointed out that the architecture itself does not impose any limitations on testing host-based intrusion detection algorithms. If a sensor for processing operating system audit trails is provided, both types of algorithms can be tested.

Simulators consist of real simulators and virtual simulators. Real simulators are programs which can explore system vulnerabilities and launch attacks. Virtual simulators can generate a series of events representing a certain type of attacks. A virtual simulator should be used only when the real simulation is not applicable, e.g., the attack can destroy the testing environment or disturb normal users activity. The development of simulators is based on the classification of well-known attacks. In the current stage, we only developed several example simulators for each class of attacks.

The event buffer is a central repository where all events are stored, including those generated by the sensor and those produced by the virtual simulators. The event buffer functions as the event pool in the model of multiple producers and multiple consumers. The sensor and virtual simulators, which generate events, are producers. Intrusion detection algorithms, which retrieve events, are consumers. To ensure that the analysis results from different detection algorithms are comparable, all detection algorithms should have a consistent view of events. This is accomplished by keeping an event in the buffer until it is consumed by all detection algorithms. To improve system performance and extensibility, each producer or consumer runs as an individual process in the operating system. To achieve maximum efficiency of interprocess communication, the event buffer is implemented by shared memory. Processes are synchronized using semaphores.

The visualization module can be used to manage the platform, for example, configure the event buffer, start and stop simulators, etc. It can also present the analysis results of detection algorithms in an intuitive way. For example, it will print a message in the message panel when a communication path is found to be anomalous or in violation of security policy. It can also visualize the traffic hierarchy constructed by a sample detection algorithm we have implemented.

To test and prove the applicability of the architecture, we implemented a network intrusion detection algorithm and tested it with the platform. The algorithm is based on the idea of Network Security Monitor (NSM) [13] developed at University of California at Davis. It is composed by an event assembler and two analysis functions, statistical analysis and signature analysis.

Event assembler is responsible for restoring network events into TCP sessions and organizing sessions into a hierarchical structure. Analysis functions are called periodically to check the security status of each TCP session and report insecure sessions to the visualization module. Statistical analysis looks for unusual traffic patterns. This requires the knowledge of historical traffic patterns. We collected three weeks worth of network traffic from a subnet at our department, and built historical profiles for normal traffic patterns. The lower the probability of a communication path in historical profiles, the more anomalous that communication path is. For example, if a TELNET session from machine A to machine B is very rare, the occurrence of such a connection may indicate an intrusion. Signature analysis is to look for a specific traffic pattern which indicates a known intrusion or a violation of security rules. For example, all accesses to machine A are restricted to within the organization. Any connection to machine A from outside is a violation of security rules and will fire an alarm.

4 Experiments

4.1 Objectives

Our long term goal is to develop innovative intrusion detection algorithms and test them with IntruDetector. As the development of IntruDetector is still undergoing, we only present the preliminary results of our experiments here. The objective is to demonstrate the capability of the software platform and how to use the platform to perform testing. We study the behaviors of an intrusion detection algorithm we have implemented. Both real simulation and virtual simulation are applied.

4.2 Experimental Environment

We conducted the experiments in a real environment, which is depicted by Figure 2. The experimental network consists of a server and several clients, and is directly connected to the Internet without firewall protection. There were tens of real users who were using several common network services (e.g., mail, ftp, X windows, among others) on a daily basis. This is a typical environment seen by many organizations.

4.3 Cases Selection

The following test cases are selected.

- Case 1. An FTP session from a client to the server. The session is terminated after a sensitive

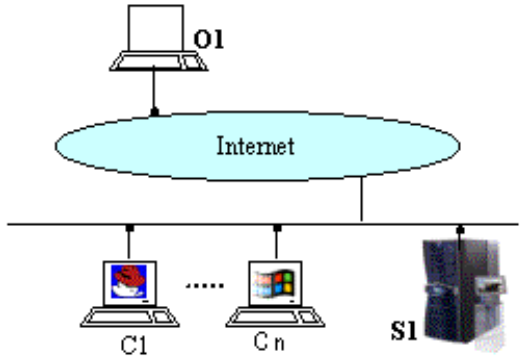

Figure 2: The Experimental Network

file and other documents are downloaded. This session is generated by real simulation.

- Case 2. A telnet session from a outside machine to the server at midnight. It is probably a masquerader since it is unlikely that a user will login at midnight from the outside in our environment. This session is generated by virtual simulation. Actually, it is very difficult to be produced by real simulation since it requires us to have access to the outside machine.

- Case 3. SYN-flooding denial-of-service attack on the server. To avoid being detected, the source of SYN messages is set to the local machine. The service is set to the most commonly used service, X11. This attack is simulated by virtual simulation. It is possible but dangerous to emulate the attack by real simulation.

4.4 Statistical Analysis Results

We collected three weeks worth of data from the experimental environment. The data consist of 16922 TCP connections. Figure 3 shows the features of several common TCP services in the historical data. We can see the number of HTTP connections is the greatest. From the perspective of the amount of data transmitted over a connection, X11 consumes more network bandwidth than others as the average number of packets or bytes of data transmitted over a X11 connection is the greatest.

The distribution of the services used by machine *C1* to communicate with Server *S1* is depicted in Figure 4. We can see that X11 is used most often. FTP is also frequently used.

Service	Number of Connections	Average Number of Packets	Average Number of Bytes
ftp	261	161	145,806
telnet	101	691	9,770
smtp	98	41	9,155
finger	6	7	45
http	10,375	12	3,332
rpc	786	8	134
login	195	663	15,053
printer	79	149	197,835
X11	169	26,986	1,089,006

Figure 3: The features of Common TCP services in Historical Profile

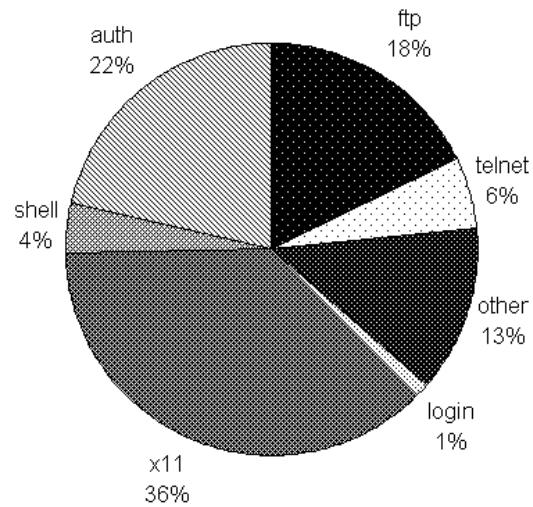

Figure 4: Distribution of the Services from Client *C1* to Server *S1*

The Results of statistical analysis are presented as follows:

- Case 1. This session is not flagged as suspicious since the probability of the occurrence of the session is higher than the predefined threshold. However, the session is actually very suspicious. It downloaded sensitive system files, and might use them to crack user passwords. Since the session was initiated from a local machine, it could be done by an internal user. It is also possible that machine was cracked and being used as the basis to launch attacks against others. The failure to identify this suspicious activity shows the weakness of statistical analysis for detecting inside misuser.

- Case 2. This session is flagged as anomalous because it did not occur before. Although this session is suspicious, it is also possible that the session was initiated by an internal user who was on

out-of-town business at that time. Flagging this session as intrusive indicates that statistical analysis may trigger many false alarms. This is due to the immaturity of the detection algorithm we implemented. If the algorithm can adapt to the change of user behaviors, false alarms may be reduced.

- Case 3. This session is not flagged as suspicious since HTTP is the most often used protocol in the experimental network. Figure 5 shows that 63 percent of connections are HTTP connections. The inability to identify SYN-flooding attack is due to the simplicity of the history profile. If the profile contains more information, such as the intensity of connections, it would be possible to detect denial of service attacks like SYN-flooding attack.

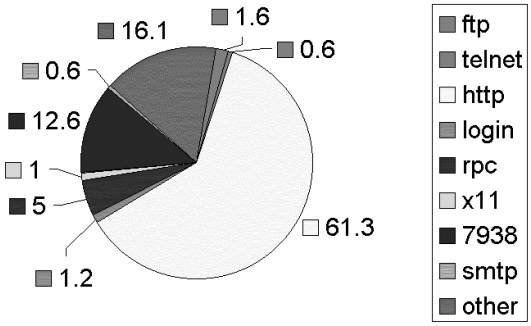

Figure 5: Distribution of Common Network Service

4.5 Signature Analysis Results

To perform signature analysis, security rules should be defined. Security rules are site dependent. They are defined by a security officer according to the organization policies of how the computing resources should be used. In our experiments, we define the following security rules. Any violation of these rules is deemed as intrusive.

- Access to the server is restricted to the inside.
- The maximum number of concurrent connections to the server is 1000.

The results of signature analysis are presented as follows:

- Case 1. The FTP session is not flagged as an intrusion since it does not violate any security rules. Although downloading sensitive files is abnormal, it can not be detected. This is due to the simplicity of the sensor in that only the information contained in packet headers are utilized. It is impossible to tell which files have been accessed without parsing the data part of packets.
- Case 2. The telnet session is flagged as intrusive as it explicitly violate rule 1.
- Case 3. Some of TCP SYN events are flagged as intrusions. Signature analysis only checks if the security rules are violated. When we began to emulate SYN-flooding attack, the total number of current connections was less than 1000. When the limit was reached, all of the newly established connections were flagged as intrusive. Although some malicious connections were found, many normal activities were also flagged as intrusions. There were many false alarms. To effectively detect this type of attacks, a simple rule is insufficient, and sophisticated techniques must be applied

5 Conclusion and Future Work

Intrusion Detection Systems (IDSs) play an important role in protecting information infrastructure. Most of the current detection algorithms are not sufficiently mature and extensive test and further improvement are required. We have developed a testbed for network intrusion detection algorithms. It allows for tests to be done in a real environment with a wide range of intrusive cases but without any danger of compromising normal system operations. Our future work will focus on the development of new intrusive simulators. We also plan to redesign our system to make it conform to the standards when they are finalized by IDWG [10].

References

[1] "Yahoo! Under Attack: A Coordinated Attack Shuts Down Leading Web Site for Three Hours". http://www.abcnews.go.com/sections/tech/DailyNews/yahoo000207.html
[2] http://www.cert.org/advisories/
[3] "White House: Hackers hit our Web site". http://www.cnn.com/TECH/computing/9905/12/
[4] http://www.ll.mit.edu/IST/ideval/index.html
[5] R. Anderson and A. Khattak. "The Use of Information Retrieval Techniques for Intrusion Detection", In *Proceedings of RAID'98*, Louvain-la-Neuve, Belgium, September 1998.

[6] Nirupama Bulusu, Jeremy Elson, Deborah Estrin, Ramesh Govindan, John Heidemann, Chalermek Intanagonwiwat, Nader Salehi, Kun-chan Lan, Ya Xu, and Wei Ye. "Effects of Detail in Wireless Network Simulation", *USC/ISI TR-20000-523*.

[7] M. Crosbie and E. Spafford. "Defending a Computer System using Autonomous Agents", *Technical Report No. 95-022*, COAST Laboratory, Department of Computer Sciences, Purdue University, March 1994.

[8] H. Debar, M. Dacier, A. Wespi, and S. Lampart. "An Experimentation Workbench for Intrusion Detection Systems", *Research Report RZ 2998*, IBM Research Division, Zurich Research Laboratory, 8803 Ruschlikon, Switzerland, March 1998.

[9] R. Durst, T. Champion, B. Witten, E. Miller, and L. Spagnuolo. "Testing and Evaluating Computer Intrusion Detection Systems", *Communications of the ACM*, July 1999.

[10] M. Erlinger and S. Chen. "Intrusion Detection Exchange Format", http://www.ietf.org/html.charters/idwg-charter.html

[11] J. Frank. "Artificial Intelligence and Intrusion Detection: Current and Future Directions", In *Proceedings of the 17th National Computer Security Conference*, October 1994.

[12] A. Ghosh and A. Schwartzbard. "A Study in Using Neural Networks for Anomaly and Misuse Detection", In *Proceedings of the 8th USENIX Security Symposium*, Washington, D.C., USA, August 23-26, 1999.

[13] T. Heberlein, G. Dias, K. Levit, B. Mukherjee, J. Wood, and D. Wolber. "A Network Security Monitor", In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 296-304.

[14] K. Jackson, D. Dubois, and C. Stallings. "An Expert System Application for Network Intrusion Detection", In *Proceedings of the 14th National Computer Security Conference*, pages 215-225, Washington, D.C., October 1-4, 1991.

[15] W. Jansen, P. Mell, T. Karygiannis, and D. Marks. "Applying Mobile Agents to Intrusion Detection and Response", *Interim Report-6416*, Computer Security Division, National Institute of Standards and Technology, October 1999.

[16] J. Bonifacio, A. Cansian, A. Carvalho, and E. Moreira. "Neural Networks Applied in Intrusion Detection Systems", In *Proceedings of the IEEE World Congress on Computational Intelligence* , WCCI'98, Anchorage, USA.

[17] S. Kumar and E. Spafford. "An Application of Pattern Matching in Intrusion Detection", *Technical Report CSD-TR-94-013*, The COAST Project, Department of Computer Sciences, Purdue University, West Lafayette, IN, USA, June 17 1994.

[18] W. Lee. "A Datamining Framework for Building Intrusion Detection Models", In *IEEE Symposium on Security and Privacy*, pages 120-132, Berkeley, California, May 1999.

[19] R. Lippmann, and I. Graf. "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation", In *Proceedings of DARPA Information Survivability Conference and Exposition*, DISCEX'00, Jan 25-27, Hilton Head, SC, 2000.

[20] S. McCanne, C. Leres, and V. Jacobson. "Libpcap", available via anonymous ftp to ftp.ee.lbl.gov, 1994.

[21] B. Mukherjee, L. Heberlein, and K. Levitt. "Network Intrusion Detection", *IEEE Network*, May/June 1994.

[22] N. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. Olsson. "A Methodology for Testing Intrusion Detection Systems", *IEEE Transactions on Software Engineering*, Vol 22, No. 10, Oct 1996, pp. 719 - 729.

[23] N. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. Olsson. " A software Platform for Testing Intrusion Detection Systems", *IEEE Software*, 1997.

[24] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni. " Analysis of a Denial of Service Attack on TCP ", In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208-223, May 1997.