Statistical Anomaly Detection for Link-State Routing Protocols *

Diheng Qu, Brian M. Vetter
Feiyi Wang, Ravindra Narayan, S. Felix Wu
Computer Science Department
North Carolina State University
Raleigh, NC 27695
{dqu, wu}@eos.ncsu.edu

Y. Frank Jou
Fengmin Gong, Chandru Sargor
Advanced Networking Research
MCNC
RTP, NC 27709
{jou}@mcnc.org

Abstract

The JiNao project at MCNC/NCSU focuses on detecting intrusions, especially insider attacks, against OSPF (Open Shortest Path First) routing protocol. This paper presents the implementation and experiments of the JiNao's statistical intrusion detection module. Our implementation is based upon the algorithm developed in SRI's NIDES (Next-generation Intrusion Detection Expert System) project. Some modifications and improvements to NIDES/STAT are made for a more effective implementation in our environment. Also, three OSPF insider attacks (e.g., `maxseq`, `maxage`, *and* `seq++` *attacks) have been developed for evaluating the efficacy of detecting capability. The experiments were conducted on two different network routing testbeds. The results indicate that the proposed statistical mechanism is very effective in detecting these routing protocol attacks.*

1 Introduction

The ever growing usage of the Internet has brought with it an increased reliance on the network infrastructure which makes it all possible. Routing (e.g., RIPv2, EIGRP, BGP, and OSPFv2) and network management (e.g., SNMPv3/ng) protocols form the core heart of this infrastructure. Until recently, the security of these protocols has not been fully emphasized. However, there is a growing awareness of the potential consequences of attacks aimed the infrastructure, particularly the routing protocols.

The JiNao project [3] at MCNC/NCSU focuses on detecting intrusions, especially insider attacks, against network routing protocols. Here,"insider" refers to a trusted entity participating the routing information exchange process or an outsider with the capability to intercept and modify the information exchange channels. Many insider attacks for link state routing protocols have been mentioned or discovered [5]. For example, in [8], we have discovered and implemented a new OSPF insider attack which allows the attacker to control the network topology for up to one hour by injecting a small number of bad OSPF PDUs. This attack is due to an implementation bug on many commercial routers. One particular major router vender has recently responded to our discovery by giving us a new version of their router software which makes the attack less effective.

One approach to defend routers against insider attacks is to depend on intrusion detection systems (IDS). The IDS approach may be relatively more acceptable to the industry as it requires no changes to the routing protocols themselves. The JiNao system currently under development takes this intrusion detection approach to handle insider attacks.In this paper, we limit our discussion to the *statistical intrusion detection mechanisms* for protecting link-state routing protocols. We have applied the statistical ID algorithm [2, 7] developed in SRI's NIDES (Next-generation Intrusion Detection Expert System) for detecting anomaly of link state routing protocols. Some modification and improvements are made, as will discuss later, to make the implementation more effective in our environment. To support our experiment, we implemented the NIDES algorithms for the OSPF routing protocol and three OSPF insider attacks (*maxseq, maxage,* and *seq++*), all running on both NCSU's and MCNC's routing testbed. As will be presented later, our results show that our proposed methodology is very effective in detecting these routing protocol attacks.

In the next section, we briefly introduce some background information about the NIDES/STAT algorithm. We assume that the reader has some background about OSPF [4]. Section 3 presents three insider attacks regarding their their design, implementation, and effect. The detailed discussion of the implementation of NIDES/STAT algorithm is in Sec-

*This work is supported by the U.S. Department of Defense Advanced Research Projects Agency and the U.S. Air Force Rome Laboratory under contract F30602-96-C-0325.

0-8186-8988-9/98 $10.00 © 1998 IEEE

tion 4, while the experimental results are provided in Section 5. Finally, in Section 5, we discuss our experience in using NIDES/STAT for detecting given router attacks.

2 The NIDES/STAT Algorithm

2.1 Mathematical Background

The NIDES/STAT algorithm monitors a subject's (either a user or a software program) behavior on a computer system, and raises alarm flag when the subject's current (short-term) behavior deviates significantly from its expected behavior, which is described by its long-term profile. This is achieved with a χ^2-like test [6] for comparing the similarity between the short-term and long-term profiles.

Here are some notations for the NIDES/STAT algorithm: let the current system behavior be a random variable under the sample space S. Events, $E_1, E_2, \ldots E_k$, represent a partition of S, where these k events are mutually exclusive and exhaustive. Let $p_1, p_2, \ldots p_k$ be the expected probabilities of the occurrence corresponding to events $E_1, E_2, \ldots E_k$. To verify if the random variable really has the distribution depicted by $p_1, p_2, \ldots p_k$, experiments are repeated N times independently, where N is a large number. N is also referred as *sample size*. Let Y_i represents the real number of occurrence for event E_i, we have $\sum_{i=1}^{k}(Y_i) = N$. Let p_i' represents the empirical probability for event E_i, i.e. $p_i' = Y_i/N$. Then we test the hypothesis

$$H_0 : p_i' = p_i, i = 1, 2, \ldots, k.$$

and

$$H_1 : H_0 \text{ is not ture.}$$

Let

$$Q = \sum_{i=1}^{k} \frac{(Y_i - N \times p_i)^2}{N \times p_i}$$

If the independence is assumed between events $E_i, 1 \leq i \leq k$, it has been proven that, for a large N', Q has an approximate χ^2 distribution with $(k-1)$ degrees of freedom. To get accurate approximation, it is suggested that N should be larger than 50 and $(N \times p_i)$ should be larger than 5. Otherwise, several "rare" events should be merged together to form a new event E_j such that $(N \times p_j)$ would exceed 5.

Intuitively, Q measures the "closeness" of the observed numbers to the corresponding expected numbers. If Q is "small," H_0 is accepted as a true hypothesis. If these N' experiments (where N' is a large number) are assumed independent, Q has a χ^2 distribution with $k-1$ degrees of freedom. Let q be an instance of Q. If $\Pr(Q > q) < \alpha$ (or $q > \chi^2_\alpha(k-1)$) where α is the desired significance level of the test, the hypotheses is rejected. In the context of our application, it means that the short-term profile is statistically different from its long-term profile which allows us to draw a conclusion that an anomalous behavior has occurred. Two kinds of errors are defined. *Type I error* means that the hypotheses is true but is rejected. *Type II error* means the Hypotheses is false, but is accepted. The probability that Type I error occurs is also refereed as *false positive rate*, while the probability for type II error is refereed as *false negative rate*.

In practice, however, the assumption of independence may not be true. Furthermore, there may be insufficient observations for some bins in the data stream on which Q is based. Therefore, Q may not have a χ^2 distribution. SRI's NIDES/STAT proposes a way to track the values of Q in order to establish an empirical probability distribution for Q. This distribution, along with distribution of the system's expected behavior, is saved in a long-term profile, which is updated once per day in a real-time operation.

The NIDES/STAT algorithm defines another variable S which is transformed from the tail probability of Q distribution such that S has a half-normal distribution. The purpose of defining S variable is to allow the degree of abnormality from different types of measures to be added on a comparable basis. The formula for S is:

$$S = \phi^{-1}(1 - \frac{\Pr(Q > q)}{2}),$$

where $Pr(Q > q)$ is the tail probability and ϕ is the cumulative normal distribution function of an $N(0, 1)$ variable.

2.2 Real-Time Intrusion Detection

The NIDES/STAT algorithm is capable of conducting real time intrusion detection. The statistical component receives audit data in real time, which serve both to score current behavior and to update long-term profiles. The former operation is only possible after a period of initial statistical profile training. When a new audit record is received, the statistical component applies a fading factor to the short-term profile before newly arrived record is incorporated (see next subsection for more details). This updated short-term profile is then compared against the system's long-term behavior by computing the values of Q and S. The statistical module also maintains summaries of all the activities since the last long-term profile update. At the end of the day (or any appropriate period of time for updating the long-term profile), the existing long-term profile is exponentially faded, and the summary of activities for that day being maintained by the statistical component is incorporated to produce a new long-term profile.

2.3 Weighted Sums (Effective N, N_{eff})

The sample size N dictates the time span of so called *"short-term."* Intuitively, a *"sliding window"* can be implemented to keep the most recent N pieces of audit records. Whenever a new audit record arrives, the window slides to cover it and the oldest record in the previous window is discarded. Then, $Y_i, 1 \leq i \leq k$, are updated to obtain the new values of Q and S. However, when N is too big, the "sliding window" scheme may consume too much computing resources. To deal with this problem, a *weighted sum* scheme (called effective N, or N_{eff}) was proposed in NIDES/STAT. Let τ_s be defined as a short-term fading factor. When a new event E_i occurs, all $Y_j, 1 \leq j \leq k$ are re-calculated with the following formula:

$$Y_j^{new} = \begin{cases} Y_j^{old} \times \tau_s + 1 & \text{if } j = i \\ Y_j^{old} \times \tau_s & \text{if } j \neq i \end{cases}$$

and,

$$N_{eff}^{new} = \sum_{i=1}^{k}(Y_i^{new}) = \sum_{i=1}^{k}(Y_i^{old}) \times \tau_s + 1 = N_{eff}^{old} \times \tau_s + 1.$$

By using this scheme, the calculations can be done recursively and efficiently. The computing time is proportional to k, which is the number of events. Typically, it is much smaller than the sample size N. Also this scheme saves lots of memory since it only requires to remember k variables rather than N records. It can be easily derived that N_{eff} has an asymptotic value of $\frac{1}{(1-\tau_s)}$.

2.4 Long-term Profile Training

Training is the process by which the statistical component learns normal behavior for a subject. In NIDES/STAT, the profile training consists of three phases:

Category, C-Training: to learn the subject's expected behavior, *i.e.,* probabilities for events, E_i;

Q Statistics, Q-Training: to learn empirical distribution for Q statistic which measures the deviation between short-term observations and long-term expected category distributions;

Threshold, T-Training: wherein the system establishes the threshold for the measures.

After the training, by updating long-term profile at a regular interval (for example, once per day), the algorithm allows adaptation to graduate changes of a subject's behavior. A long-term fading factor τ_l is defined so that the profile will "forget" the ancient data gradually. On the other hand, if the system behavior changed abruptly, *e.g.,* system upgrade, the statistical component must provide a way to learn this change more quickly. For example, it may discard the old long-term profile, and go through the three phases training again.

2.5 Apply Partition

NIDES defines four classes of measures: (1) *activity intensity measure*, which measure whether the volume of activity generated is normal; (2) *categorical measure,* whose values are by nature categorical, like OSPF packet type. (3). *continuous or counting measures*, whose values are numeric, like LSA (Link State Advertisement) age or CPU usage, and (4) *audit record distribution.* We found the first three classes useful for detecting routing protocol attacks. And, we made some modifications when dealing with the *activity intensity measure.* We used the same formula (please refer to Section 4.1) to map the volume of the system's audit data to a floating point number. Then, we treat it as a *counting measure*, instead of the approach proposed in NIDES [2].

3 OSPF Attacks

In order to validate the propose statistical approach, we have implemented three OSPF insider attacks for the FreeBSD platform.

3.1 Attack 1: Simple Modification

When the attacker receives a LSA, it can modify the link state metric and increase the LSA sequence number by 1 (*i.e.,* Seq++). The attacker also needs to re-compute both the LSA and OSPF checksums before the tampered LSA is re-injected into the system. This attack LSA, because it has a bigger LSA sequence number, will be considered "fresher" by other routers. And, eventually it will be propagated to the originator of this particular LSA. The originator, according to the OSPFv2 specification, will "fight-back" with a new LSA carrying correct link status information and an even fresher sequence number.

3.2 Attack 2: Max Age

When the attacker receives a LSA, it can modify the LSA age to `MaxAge` (*i.e.,* 1 hour), and then re-compute only the OSPF checksum before the tampered LSA is re-injected into the system. This attack LSA, with the same sequence number but `MaxAge`, will cause all routers to purge the corresponding LSA from their topology database. Eventually, the originator of this purged LSA will also receive the MaxAge LSA. The originator, according to the OSPFv2 specification, will "fight-back" with a new LSA carrying correct link status information and a fresher sequence number.

3.3 Attack 3: Max Sequence Number

When the attacker receives a LSA, it can modify the link state metric and set the LSA sequence number to `0x7FFFFFFF` (*i.e.,* MaxSequenceNumber). The attacker also needs to re-compute both the LSA and OSPF checksums before the tampered LSA is re-injected into the system. This attack LSA, because it has the biggest LSA sequence number, will be considered the "freshest" by other routers. And, eventually it will be propagated to the originator of this particular LSA. The originator, according to the OSPFv2 specification, "should" first purge the LSA (setting MaxAge) and then flood a new LSA carrying correct link status information and the smallest sequence number: `0x80000001`.

4 Statistical Measures

The selection of statistical measures should base on good understanding about the system itself as well as all possible attacks that may influence the system's normal behavior. On one hand, we can choose a large number of different statistical measures such that the statistical IDS would be very sensitive to intrusions. This implies that we, simultaneously, monitor many different aspects of the system's behavior. On the other hand, a real-time intrusion detection system should not introduce too much performance overhead.

4.1 Data Volume

The first measure we chose is data volume which is one of the activity intensity measures. This measure monitors the inter-arrival time of all the OSPF packets which a router receives. In a stable routing domain, routers exchange routing information regularly, the volume of the traffic tends to have a fixed pattern. As we mentioned earlier, attacks may invoke "fight back", which will cause additional routing traffic. In fact most anomalies tend to manifest their abnormal behaviors through data volume variation. Therefore, this measure is useful in detecting these anomalies.

The data volume is represented by a floating point number, R_n, which is calculated by following formula:

$$R_n = R_{n-1} \times 2^{(-\tau_f \times \delta T)} + 1$$

where τ_f is a predefined constant, and δT is the event inter arrival time. In Section 6.1 we will discuss how to determine the sample space for R_n, which is cut into 32 bins on a geometric scale.

4.2 OSPF Type

There are 5 types of OSPF packets: *(1) Hello, (2) Database Description, (3) Link State Request, (4) Update* and *(5) Acknowledge.* In a routing domain, each OSPF router will send Hello packets to its adjacent neighbors at a fixed interval. Type 2 and 3 OSPF packets are used only when a new adjacency is initialized. Update OSPF packets (type 4) are sent out by each router at a same interval to maintain the freshness of its LSAs. Each updated LSA has to be acknowledged. But a router may choose to delay the acknowledgment, so that it can put several LSA acknowledgments in one OSPF packet later to save some bandwidth. Under a normal condition, the distribution of OSPF packet type is very stable. If some attack occurs, it may be disturbed.

4.3 LSA Age

In OSPF, each LSA has an "age" field in its header describing how long this LSA has been living. When an LSA travels through the network, the intermedia routers will change this field, adding transmission delay and processing delay to it. An LSA is also aged regularly when it is in a router's database. OSPF protocol specifies that if an LSA's age is larger than one hour, it must be discarded. If any intermedia router is comprised, it may modify this field to a larger number to disturb down-stream routers. Also, like the mutable TTL (Time To Live) field in IP header, the "age" field is not included when calculating LSA checksum. So it's easy for an intruder to modify this field without being detected. This measure is chosen specifically to protect this weak point.

LSA "age" is a counting measure. The possible value is from 0 to 3600 (in the unit of second, 3600 seconds = 1 hour). The space is lineally cut into 30 bins with 120 seconds per bin.

5 Experiment Results

In this section, we will present our experimental results in applying the statistical intrusion detection mechanism to handle various network router insider attacks. We will first evaluate the false positive rate. Then, we will show some testing results for three attacks, simple modification attack, max sequence attack, and max age attack. Please note that the results are either **Red Alarm**, **Yellow Alarm**, or **Normal**, which represent "$<$ 0.5%" (tail probability of Q distribution), "between 0.5% and 5%", or "$>$ 5%" respectively.

5.1 Testbeds

We have set up two testbeds for the JiNao project, one in MCNC and the other in NCSU. The topology of these experimental testbeds are shown in Figure 1.

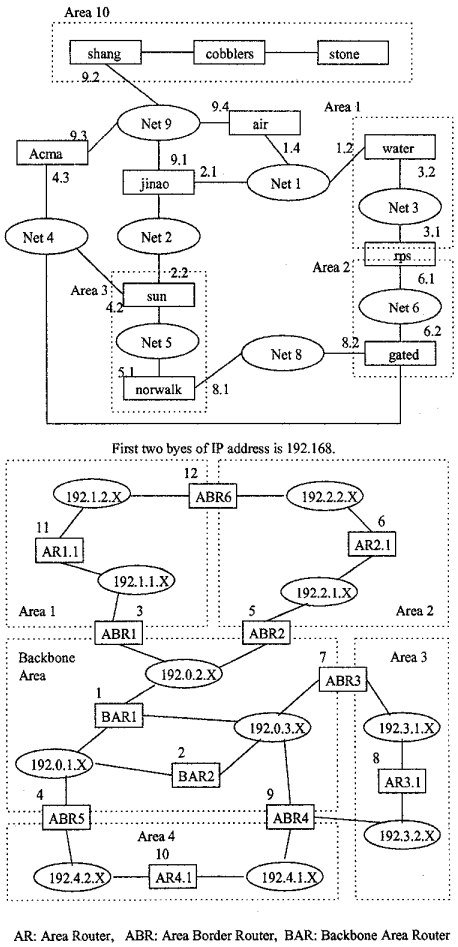

Figure 1. Topology of NCSU/MCNC testbeds

5.2 Evaluation of False Positive Rate, fp

We define false positive rate as the probability that the statistical component will raise alarm flag while under normal condition. We evaluated the value on the base of a one-day period, using the following equation:

$$fp = \frac{number\ of\ alarm}{The\ total\ number\ of\ observation\ in\ a\ day}$$

Table 1 shows the average results that we have collected from the two testbeds.

The high false positive rate for data volume measure is caused by the way we calculate R_n,

$$R_n = R_{n-1} \times 2^{(\tau_f \times \delta T)} + 1;$$

Since routers do not have a globe synchronized clock, they generate events according their own timers. Then the interval time between two events which are generated by two different routers is changing all the time, i.e.δT is drifting. This makes R_n unstable. Sometimes, if R_n is near the boundary of two categories, a small change in R_n will make it jump from one category to another. This will result in high false positive rate.

Routers	data volume	OSPF packet type	LSA age
shang	5.18%	0.00%	0.00%
norwalk	19.87%	0.00%	0.00%
BAR2	0.00%	0.00%	0.00%
AR4.1	0.00%	0.00%	0.00%
ABR3	22.46%	0.00%	0.00%
ABR4	0.00%	0.00%	0.00%

Table 1. Average False Positive Rate

5.3 Test Results for Simple Modification Attack

The attacker, A, chooses one particular router, B, as the victim. Whenever it receives a LSA originated by B, A will modify the "cost" field (called $metric$ in LSA) to the largest value, which means the link is down, and broadcast it other routers. This is going to cause "fight back", which we mentioned before.

The following results are collected from MCNC's testbed. We run the statistical module on four different routers, BAR2, AR4.1, ABR3, ABR4 (Please refer to Figure 1 for their locations). The attacker resides on ABR5. In the tables, "Target" means the victim B, N means the number of attacks, i.e. after modifies N LSAs, the attacker will quit.

Routers	data volume	OSPF packet type	LSA age
BAR2	Red Alarm	Red Alarm	Normal
AR4.1	Normal	Normal	Normal
ABR3	Red Alarm	Red Alarm	Normal
ABR4	Red Alarm	Red Alarm	Normal

Table 2. N= 15, Target : ABR2

Table 2 shows the result when we attacked ABR2 from ABR5. The attacker ran an interception module on ABR5 to intercept all the OSPF packets before they enter ABR5's OSPF protocol engine. If it was not interested in a packet, the attacker would just let it go (to ABR5's routing daemon). But if it found an OSPF update packet containing an LSA which is originated by ABR2 to advertise its links, the attacker would change the link state from up to down, increase the sequence number by 1, re-compute the checksum,

and forward it to ABR5. In this case, ABR5 was tricked to think: oh, ABR2's links are down. It put the LSA into its database, and sent an acknowledge packet back to BAR1. BAR1 was waiting for the acknowledgment for the correct LSA. By comparing the headers, BAR1 found the ABR5's ack was not what it expected. So it re-sent the correct LSA to ABR5 again. This time, the attacker didn't do anything, just let it go. Now, ABR5 thought: my database has a newer version of this LSA (since its sequence number is higher), but BAR1 still has the older one, I have to send him my copy to make our database consistent. Then BAR1 got the newer copy, which in fact was created by the attacker. It forwarded it further to all its neighbors. Soon everyone, ABR1, ABR2, ABR3, ABR4 and BAR2, got the bad LSA. ABR2, which is the legal originator, fought back. The attacker did the same thing for 15 times, then quit. All the routers in the backbone area observed the abnormal traffic, while router AR4.1, which is in Area 4, hadn't been disturbed. So in table 2, we can see that AR4.1 didn't raise any alarm. All other routers detected the anomaly by the data volume measure and OSPF packet type measure. Because the fight back brought additional routing traffic, especially additional OSPF update/ack packets.

Table 3 shows the result for another similar attack. The only difference is that router ABR3 was the victim this time.

Routers	data volume	OSPF packet type	LSA age
BAR2	Red Alarm	Red Alarm	Normal
AR4.1	Normal	Normal	Normal
ABR3	Red Alarm	Red Alarm	Normal
ABR4	Red Alarm	Red Alarm	Normal

Table 3. N= 15, Target : ABR3

5.4 Test Results for Max Sequence Attack

We ran the statistical module on BAR2, AR4.1, ABR3, and ABR4. But this time the attacker resides on ABR2, and the victim is ABR5. Table 4 shows the result of the max sequence attack.

Routers	data volume	OSPF packet type	LSA age
BAR2	Red Alarm	Red Alarm	Normal
AR4.1	Normal	Normal	Normal
ABR3	Normal	Normal	Normal
ABR4	Normal	Normal	Normal

Table 4. Max sequence attack

As mentioned before, because of the implementation bug, the routing daemon process can not handle max sequence attack correctly. The victim will keep fighting back with its neighbor. So, BAR2 has detected the anomaly, since it is adjacent to ABR5. But all other routers did not know that something has been going wrong. This also implies that to protect a routing domain, one need to run Jiano on several routers to increase the odds to detect intrusions. This also confirms our suggestion in [8] that routing protocol design and implementation should avoid *hit-and-run* attacks as much as possible.

Please note we also ran the max sequence attack against a commercial router in NCSU's testbed. With the original version of routing software, the commercial router is vulnerable to the max sequence number attack. However, after we contacted the vender of this particular router, they have responded to our discovery a few months ago by giving us a new version of their router software which has fixed the bug. With the fixed version, the max sequence attack caused a global fight-back in backbone area (just like the simple modification attack). When the attacker only launched one max sequence attack, the bad LSA was purged out (by the originator's fight-back) in a few seconds. If the attacker kept attacking, all the JiNao module in the backbone area reported the anomaly.

5.5 Test results for Max Age Attack

We ran the statistical module on ABR2, ABR3, ABR4, ABR5 and BAR2. The attacker was on BAR1 and launched 10 max age attacks. The victim is ABR2, and Table 5 shows the result.

Routers	data volume	OSPF packet type	LSA age
ABR2	Red Alarm	Normal	Red Alarm
ABR3	Red Alarm	Normal	Red Alarm
ABR4	Red Alarm	Normal	Red Alarm
ABR5	Red Alarm	Normal	Red Alarm
BAR2	Red Alarm	Normal	Red Alarm

Table 5. Max age attack

5.6 Other Abnormal Situations

Statistical Module will raise alarm flag whenever system's current behavior deviates too much from its historical behavior. In certain cases, alarm flags may mean anomaly rather than attack. For example, when a router is restarted, the normal routing traffic pattern will be disturbed while the router brings up its adjacencies by exchanging information with its neighbors. The statistical components should be able to detect this kind of anomaly.

Routers	data volume	OSPF packet type	LSA age
BAR2	Red Alarm	Red Alarm	Red Alarm
AR4.1	Normal	Normal	Normal
ABR3	Red Alarm	Red Alarm	Red Alarm
ABR4	Red Alarm	Red Alarm	Red Alarm

Table 6. Restart router ABR1

Table 6 shows the result when router ABR1 was restarted (the routing daemon was brought down first, then brought up immediately), it tried to re-establish the adjacency relationship with its neighbors and sent them LSA to advertise its links. The sequence number was reset to `0x80000001`. But the outside world still had the LSA which ABR1 advertised before it "crashed". Though those copies were older, they had higher sequence numbers. Hence ABR1 had to send out max age LSAs to purge the older copies from the other routers' database. This caused anomaly not only in data volume and OSPF packet type, but also in LSA age because of the max age LSAs.

6 Experience and Improvement

In the following sections, we present some experiences gained and improvements made in the implementation of NIDES/STAT algorithm.

6.1 How to Deal with Counting and Intensity Measure

6.1.1 Max Value

As mentioned in Sections 2.5 and 4, we treat the intensity measure the same as a counting measure. One common property shared by these two classes of measures is that the behavior are represented by a number, R_n, which can be either integer or floating point number. Counting measures are transformed to categorical measures by cutting R_n's range into several bins to constitute a set of partitioned events. An interesting issue here is how to determine the possible range of R_n. One need to set an upper bound, R_{Max}, for R_n so that $Prob(R_n > R_{Max}) < threshold$. For some counting measures, it's trivial to set the upper bound. For instance, the range for the LSA age is an integer number from 0 to 3600 (seconds). But for data volume, it depends on the number of neighbors and the size of the routing domain. So R_{Max} can not be hard coded in the program.

Our module provides a way to find out what the R_{Max} should be. In the training phase, a sub-state is defined to collect R_n's mean value and standard deviation. R_{Max} is calculated by adding R_n's mean value to four times of its standard deviation. This scheme did well in our testing. Also we found that the implementation is not very sensitive to R_{Max}. But a too high or too low upper bound will harm the detecting effectiveness.

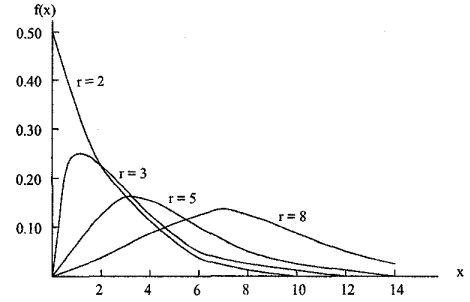

Figure 2. chi-square distribution

6.1.2 Number of Bins

Another problem is how to define N_b, which is the total number of bins. In NIDES algorithm, a magic number 32 is chosen. But we felt it's better for the user to make the decision, while given 32 as the default value.

There are some trade-offs when choosing N_b. If N_b is too small, the bins are too "coarse". In some cases, one may find most values reside in the one or two bins, which results in insensitivity to intrusions. On the other hand, if N_b is too big, the bins tend to be too "fine", so that lots of them would be associated with very small probabilities. Not only does it require more system resources for book-keeping and updating due to the pre-condition: $N_{eff} \times p_i > 5$, it will also introduce extra overhead to merge those rare categories.

Another performance penalty will be introduced if N_b is too big. As a simple example, let's assume independence between all these bins. Then, Q's distribution will be χ^2 with degrees of freedom $(N_b - 1)$. The larger the degrees of freedom is, the fatter the chi-spare curve will be, as shown in Fig. 2. This leaves the Q's distribution very hard to track and handle.

6.2 How to Determine Half-Life Parameters and the Window Size.

Due to the nature of the routing protocol, the measures do not behave in a random fashion but with a periodic pattern. The strong periodic patterns are attributed to the fact that OSPF routers exchange information periodically, like sending out Hello packet every 10 seconds for Ethernet, and refreshing old LSA every half an hour. Since NIDES algorithm computes average probabilities, in order to obtain sensible results the window size should be multiplies of the period. If this can't be achieved, the window size must

be long enough to cover several periods, so that when the window is sliding, the average probabilities would not be changed dramatically. For instance, we assume to have two events E_1 and E_2, with $p(E_1) = 0.5, p(E_2) = 0.5$. Experiments yield a stream of events: $E1$, $E1$, $E1$, $E2$, $E2$, $E2$, $E1$, $E1$, $E1$, $E2$, $E2$, $E2$, ... Table 7 gives the relationship between window size and the "real" observations of average probabilities.

Window Size	p(E_1)' min/max	p(E_2)' max/min
6, 12, 18, ...	0.50/0.50	0.50/0.50
3	0.00/1.00	1.00/0.00
9	0.33/0.67	0.67/0.33
15	0.40/0.60	0.60/0.40
21	0.43/0.57	0.57/0.43

Table 7. Window size and Average Probabilities

6.3 How to Train the Long-Term Profile

As mentioned in section 2, there are three phases to train the long-term profile. How to determine the training periods for these three phases is not a trivial task. Some experience are given below according to our experiments. The requirement for C (category) training is somehow not as critical as that for the other two training process. If this training period is not long enough, the error will be big for those p_i's (refer to section 2.1 for the definition of p_i) with small values. But remember, in calculating Q, we have a pre-condition : $N \times p_i > 5$. If this does not hold, merge will need to take place. Since those events whose Y_i's are small tend to be merged together, it therefore reduce the necessity to get those small p_i's precisely.

But in the Q training phase, even small error in the Q's distribution will potentially have an effect in the calculation : $P = Prob(Q > q)$, which can be rewritten as :

$$P = \int_q^\infty f_Q(x)dx.$$

in which $f_Q(x)$ is Q's probability density function

The function $f_Q(x)$ is depicted in Figure 3. If the training period is not long enough, one will not get the long tail part of $f_Q(x)$ correctly because of lack of occurrence of big Q value, so the the curve tends to be the one drawn in dashed line. The threshold, a, has been moved left hand.

To solve this problem, we defined a state called half functional state, which can be entered after a measure finishes its C(category) training phase. Theoretically, a measure in this state is not ready for intrusion detection, since Q's

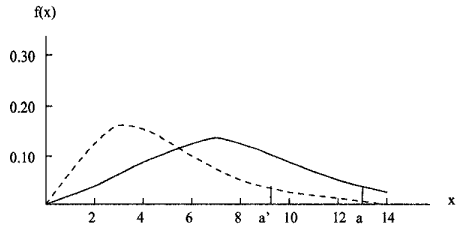

Figure 3. Q training

distribution is unknown. But in practice, we temporarily assume independence between categories, *i.e.*, Q has a χ^2 distribution, and use chi-square test to compare system's current behavior with its expected behavior. At the same time, Q's real distribution is being established, so that the measure may transit to full functional state at proper time. This scheme did very well in our testing.

Of course, our tests are not exhaustive. It is possible that for some measures, one has to wait until Q's distribution has been fully established before proceeds with any intrusion detection.

6.4 How to Provide Adequate Information to Help Conduct Inquiries?

Due to the nature of statistical approach, false positive alarm will be raised even without intrusion activity. A good deal of the system security officer's time will therefore be devoted to investigating what causes the false alert. The statistical procedure must provide the security officer with understandable information that can be used to identify the reason of an alarm in a timely fashion.

In our module, whenever an alarm is raised, the module provides several pieces of information. First is the distributions : the expected one and the observed one. Table 8 gives such an example. One can see that the number of Type 4 packets is about 34% more than expected.

OSPF Type	Expected Dist.	Observed Dist.
Type 1	0.9307	0.9158
Type 2	0.0000	0.0000
Type 3	0.0000	0.0000
Type 4	0.0410	0.0549
Type 5	0.0283	0.0293

Table 8. Difference in distributions

Second, each measure has a queue (in fact, it is implemented as a hash table) associated with each event. For measures dealing with OSPF packets, like data volume measure and OSPF packet type measure, the queue saves (OSPF

header, number of occurrence) pairs using OSPF header as the key. When compare two OSPF headers, only two fields are considered, one is OSPF router ID, the other is OSPF area ID. For measures dealing with LSA, like LSA age measure or those dynamically added LSA measures, the queue saves the (LSA header, number of occurrence) pairs in it. When one conducts comparison, only LS type, link state ID and advertising router ID are taken into consideration. Again taking OSPF packet type measure as an example, when a type i OSPF packet is received (*i.e.,* event E_i occurs), the header of the packet will be logged into the queue associated with E_i. If the header is already in the queue, increase the "number of occurrence", if not, add a new entry to it. Armed with this information, the security officer could quickly identify which router cause the problem and conduct further investigation.

For example, in Section 5.6, Table 6 shows that when router ABR1 was restarted, several statistical components raised alarm flags, including the LSA age measure on router BAR2. This measure first dumped a distribution table (Table 9) like what we have seen in the previous section. From this table, we know that the anomaly was caused by observing of too many LSAs whose ages were in bin 29.

LSA Age Bin	Expected Dist.	Observed Dist.
0	0.99914	0.98535
1	0.00000	0.00000
⋮	⋮	⋮
28	0.00000	0.00000
29	0.00086	0.00907

Table 9. Difference in distributions

7 Conclusions

In this paper, we present the statistic intrusion detection approach taken by the JiNao network infrastructure protection project at MCNC/NCSU. We discuss our ideas, design, implementation, experimental results as well as the valuable experience we got. We take the NIDES/STAT algorithm as a starting point, and we have shown how to apply and extend the NIDES/STAT approach to detect intrusions in the domain of network routing protocols. Although our presentation here focuses only on OSPF, our design, implementation and experience can be ported to protect network routing protocols in general.

Through our experiments with three OSPF insider attacks: `seq++`, `maxseq`, and `maxage` running on two different routing testbed, the results show that the proposed statistical mechanism is very effective in detecting these three attacks. The attacks can be detected, in fact, even when the network testbed itself is not very stable. We implemented the NIDES/STAT algorithm for detecting OSPF anomalies. While some modifications and improvements are needed to make it work better for our system, we found that the algorithm in general is easy to understand, implement and yet effective to detect anomaly. The performance overhead it incurs in terms of disk space and computing time is rather minor.

References

[1] S. Cheung and K. Levitt. Protecting Routing Infrastructure from Denial of Service Using Cooperative Intrusion Detection. In *New Security Paradigms Workshop*, Cumbria, UK, September 1997.

[2] H. S. Javitz and A. Valdes. The NIDES Statistical Component: Description and Justification. Technical report, SRI International, March 1993.

[3] F. Jou, F. Gong, C. Sargor, S. F. Wu, and R. Cleaveland. Architecture Design of a Scalable Intrusion Detection System for the Emerging Network Infrastructure. Technical Report E296, Adavnced Network Research, MCNC, April 1997.

[4] J. Moy. OSPF Version 2. Network Working Group Request for Comments: 2178, July 1997.

[5] S. Murphy and M. Badger. Digital Signature Protection of the OSPF Routing Protocol. In *Internet Society Symposium on Network and Distributed Systems Security*, 1996.

[6] R. V. H. . E. A. Tanis. *Probability and Statistical Inference.* Macmillan Publishing Company, 4th edition, 1993.

[7] A. Valdes and D. Anderson. Statistical Methods for Computer Usage Anomaly Detection Using NIDES. Technical report, SRI International, January 1995.

[8] B. Vetter, F. Wang, and S. Wu. An Experimental Study of Insider Attacks for the OSPF Routing Protocol. In *IEEE International Conference on Network Protocols (ICNP)*, pages 293–300, October 1997.