Service Specific Anomaly Detection for Network Intrusion Detection

Christopher Krügel
Distributed Systems Group
Technical University Vienna
A-1040 Vienna, Austria
chris@infosys.tuwien.ac.at

Thomas Toth
Distributed Systems Group
Technical University Vienna
A-1040 Vienna, Austria
ttoth@infosys.tuwien.ac.at

Engin Kirda
Distributed Systems Group
Technical University Vienna
A-1040 Vienna, Austria
ek@infosys.tuwien.ac.at

Abstract

The constant increase of attacks against networks and their resources (as recently shown by the CodeRed worm) causes a necessity to protect these valuable assets. Firewalls are now a common installation to repel intrusion attempts in the first place. Intrusion detection systems (IDS), which try to detect malicious activities instead of preventing them, offer additional protection when the first defense perimeter has been penetrated. ID systems attempt to pin down attacks by comparing collected data to predefined signatures known to be malicious (signature based) or to a model of legal behavior (anomaly based).

Anomaly based systems have the advantage of being able to detect previously unknown attacks but they suffer from the difficulty to build a solid model of acceptable behavior and the high number of alarms caused by unusual but authorized activities. We present an approach that utilizes application specific knowledge of the network services that should be protected. This information helps to extend current, simple network traffic models to form an application model that allows to detect malicious content hidden in single network packets. We describe the features of our proposed model and present experimental data that underlines the efficiency of our systems.

Keywords

Intrusion Detection, Anomaly Detection, Network Security

1. INTRODUCTION

Intrusion detection systems (IDS) are security tools that are used to detect traces of malicious activities which are targeted against the network and its resources. IDS are traditionally classified as anomaly or signature based. Signature based systems [18] act similar to virus scanners and look for known, suspicious patterns in their input data. Anomaly based systems watch for deviations of actual from expected behavior and classify all 'abnormal' activities as malicious. As signature based designs compare their input to known, hostile scenarios they have the advantage of raising virtually no *false alarms* (i.e. classifying an action as malicious when in fact it is not). For the same reason, they have the significant drawback of failing to detect variations of known attacks or entirely new intrusions.

Because of the ability to detect previously unknown intrusions a number of different anomaly based systems have been proposed. Depending on their source of input data, they are divided into host based and network based designs.

Host based anomaly detection systems are among the first intrusion detection systems built. Denning [6] describes an approach that builds profiles based on login times and resources (e.g. files, programs) that users access. Simple statistical methods are used to determine whether observed user behavior conforms to the stored model. Unfortunately, this behavior can suddenly change and is usually not well predictable. As a consequence the focus was shifted from user to program behavior. The execution of a program is modeled as a set of system call sequences [8, 7] which occur during 'normal' program execution. When the observed sequences deviate from the expected behavior the program is assumed to perform something unintended, possibly because of a successful attack (e.g. buffer overflow). Other researchers use neural networks [9] and concentrate on the analysis of the input data that is passed to programs.

Network based anomaly detection systems do not concentrate on activities at hosts (e.g. users or programs) but focus on the packets that are sent over the network. Depending on the type of information that is used for performing the detection, one can distinguish between traffic and application models.

Systems that use traffic models monitor the flow of packets. The source and destination IP addresses and ports are used to determine parameters like the number of total connection arrivals in a certain period of time, the inter-arrival time between packets or the number of packets to/from a certain machine. These parameters can be used to detect port scans or denial-of-service attempts. Most current network based systems [12, 13, 2, 17] rely on traffic models to perform the bulk of their anomaly detection.

The application model attempts to incorporate application specific knowledge. Unfortunately, such models [3] are currently very simple and include mainly additional TCP header information or count the number of bytes that are exchanged in a session between client and server.

SAC 2002, Madrid, Spain
Copyright 2002 ACM 1-58113-445-2/02/03 ...$5.00.

As stated above, traffic models can be used to detect port scans and denial-of-service (DOS) attempts. Unfortunately, the situation changes when one considers Remote-to-Local attacks (R2L). The term Remote-to-Local attack has been coined during the DARPA sponsored MIT Lincoln Labs intrusion detection evaluation [10] and specifies intrusion attempts from remote users with the aim of getting unauthorized local access to the target host (typically `root`). Such attacks usually exploit a vulnerability of a service at the target machine. This is done by sending invalid input which causes a buffer overflow or an input validation error in the code running the service. The attacker sends one (or a few) carefully crafted packets including shell-code which is executed at the remote machine on behalf of the attacker to elevate his privileges. As the intruder only has to send very few packets (most of the time a single one is sufficient), it is nearly impossible for systems that use traffic models to detect such anomalies.

The table below shows the results of the top three intrusion detection systems for the four different intrusion classes used in the DARPA evaluation.

Detection Rates	Scans	DOS	U2R	R2L
Known Attacks	92%	80%	64%	80%
New Attacks	88%	22%	66%	8%

The figure shows that less than 10% of new R2L intrusion attempts have been detected. Known attacks describe intrusions that were used during the preparation phase of the evaluation and were known to the developers of the participating systems. New attacks describe intrusions that have been added for the actual evaluation. Therefore the numbers for the new attacks are more significant in determining the quality of IDSs. While probes or DOS attacks provide the attacker with additional information or degrade the performance of target machines, only R2L and U2R (User-to-Root)[1] intrusions actually compromise a machine. Therefore, it is important to reliably detect such attacks.

We present an anomaly based network IDS that focus on R2L attacks and uses service (application) specific knowledge to increase the detection rate of intrusions of this important type. We do not attempt to replace existing systems that detect port scans and denial-of-service attacks. Our aim is to provide an additional tool that reliably detects malicious payloads to enhance the overall security of the protected network.

2. SYSTEM OVERVIEW

The idea of service specific anomaly detection is to extend the application model from considering only packet header information at the network and transport layer (i.e. TCP flags in an application context) to include the application payload as well. Unfortunately, the payload of IP packets observed at a network usually varies dramatically. When the entirety of all IP packets is considered, one can usually deduce only very little information that can be used for statistical reasoning. Therefore we cannot process the payload of packets without some knowledge of the application that created them. This makes it necessary to partition the network traffic and independently analyze packets sent by different applications. By concentrating only on one type of traffic (hence the term *service specific*), statistical data with lesser variance can be collected. This allows to establish a notion of 'normal traffic' for each service.

At first glance, the vast number of different protocols seems to make our approach undesirable. Nevertheless, one should consider that it is not necessary to change the basic detection code for each new service. We use a generic backend that is responsible for the actual statistical anomaly detection. Service specific frontends extract data from the network and transform it into a format suitable for the backend. Additionally, only a few services need to be publicly accessible. By monitoring HTTP, DNS, SMTP and FTP traffic most of the services that have to be available for use by anonymous clients from the Internet are covered. The first prototype that we have developed is currently able to check DNS and HTTP traffic.

As stated above, anomaly detection systems detect intrusions by comparing observed behavior to expected behavior using certain metrics that are defined by the underlying model. The expected behavior (called *profile*) has to be defined by the user or can be automatically deduced during the *training period*. Because manual creation of expected behavior is cumbersome and error prone most systems extract profiles from training data. It is important to point out the difference between approaches that require classified data during the training period [11] (i.e. data samples that are marked explicitly as normal or malicious) to build their models and those that do not [14]. Systems that base on classified samples extract features that basically allow the data to be clustered into a normal and a malicious group. New data is then analyzed according to these selected features and mapped into one of the sets (obviously raising an alarm when it is classified as malicious). This approach has the drawback that classified data is rarely available in new environments where the IDS has to be deployed. Therefore, these systems have to utilize general models extracted from existing sample data. In changing environments, such designs may produce many false alarms or miss actual intrusion attempts. We follow the second approach that uses observed, unclassified traffic from the place where the system is installed to form a model of 'normal' behavior. Any traffic that deviates from that model is considered hostile.

Our system uses an initial, user definable training period during which it reads packets from the network. As stated above, this data is split into service specific traffic and forms the input to build our profile. After that initial phase, the system switches to detection mode in which the new traffic is compared to our application model to detect anomalies. When the environment changes dramatically resulting in too many false alarms, it is simple to update the application model by rerunning the training phase on the changed traffic.

The following section describes the features of network traffic that we use to build our profile and the metrics that is used to determine the deviation of actual, observed traffic.

3. SYSTEM DESIGN

Our anomaly detection system consists of two logical modules, the packet processing unit (PPU) and the statistical processing unit (SPU).

[1]User-to-Root attacks specify intrusions where a *local user* unauthorized elevates his permissions to become root. Such attacks are not directly visible on the network.

3.1 Packet Processing Unit (PPU)

The task of the packet processing unit is to read the network traffic and extract suitable input data for the statistical processing unit. Our anomaly detection bases on the analysis of the payload that is passed as input to the different network services. This implies that we cannot directly operate on the packet level itself as an attacker can easily distribute his malicious payload over several datagrams. Tools like `fragrouter` [16] allow to split and send data in several IP fragments or TCP packets. Therefore the statistical processing unit uses a *service request* as the basis for its analysis.

A service request is the user supplied data which is sent over the network to a certain service to perform a single task on behalf of that user. Usually, a network service is implemented as a daemon operating in an interactive mode that waits for incoming connections on a well-known port. In order to utilize its services one has to open a connection to the daemon port and provide input data. The input data basically consist of the exact type of desired service and additional parameters. The daemon parses the supplied information, processes them and returns the results. It then terminates the connection or awaits further input. We call the user supplied input sent to the daemon in such a single transaction a service request. Depending on the service and its exact type, requests can have different formats and layouts.

For a HTTP request to a webserver, the type of service can be GET, HEAD or POST and contains parameters like the URL or the type of browser the user runs. In case of DNS, a service request is usually a single packet that contains the DNS name which should be resolved or an IP address that needs to be mapped to a DNS name.

The PPU has to extract service requests from the stream of packets on the wire to pass them to the SPU. The packet processing takes place in two stages. The first one is service independent and performs generic IP and TCP stream reassembling. It passes complete UDP packets or acknowledged segments of a TCP stream to the second stage. This second stage needs service specific knowledge and has to extract single service requests from its input. It therefore requires basic understanding of the layout of requests although in most cases only very limited information is necessary. The end of a request can usually be determined by watching for an end-of-request character or character sequence (e.g. two CTRL-LF characters in a HTTP request) or by checking a length field in the request header (e.g. the number of fixed size resource records in a DNS query). The type of service is also easily determined by a string (e.g. GET in case of HTTP) or a header value (e.g. query type and class in case of DNS). The assignment of a type to a certain request is not enforced by the service protocol itself. It is possible to assign different types to requests that are considered equal by the service protocol (and by the daemon implementing the service). In the case of a HTTP request, the fact that it is a GET request may be used to perform a more detailed analysis on the URL (e.g. one can assign different types to requests that invoke cgi-programs or php-scripts). The complete service request together with its type is fed into the statistical processing unit.

3.2 Statistical Processing Unit (SPU)

The SPU is only considered with requests and their types. As stated above, the statistical properties of requests for different services can be very diverse. But even different types of requests for a single service can vary significantly. In case of HTTP traffic, standard GET requests look very similar but a POST request that transmits lots of data keyed in by a user into a HTML form may be different. Therefore, requests are divided into several groups where each group contains requests of types with similar statistical properties. The requests of each group are then analyzed independently. Currently, the division of requests has to be done manually by the developer who adds a new service (protocol) to the IDS. We plan to develop a metrics that allows the system to automatically find similar properties of different request types after the initial setup period and group them accordingly. Nevertheless, most requests for a certain service are very similar and a reasonable starting point is to divide all requests into groups according to the associated network service.

The following properties of a request are used to determine its anomaly score.

1. Type of Request
2. Length of Request
3. Payload Distribution

The anomaly score is compared to a threshold that can be manually set by the security administrator. It should be set to a value where no more than ten false alarms are reported per day[2]. In Section 4 we show that this level allows our system to detect a significant majority of attacks.

The anomaly score of a service request is the weighted sum of the three scores computed for each of the three properties enumerated above. A number of constant factors have been introduced into the formulas shown below. Most of them are used to force the scores calculated for each of the three properties above into the same order of magnitude and could be changed (scaled appropriately) without affecting our system. A few are needed to reflect our considerations accordingly.

3.2.1 *Type of Request*

Including the type of request as a property in calculating the anomaly score has the following rationale. Often, exploits that are based on buffer overflows or input validation errors use a feature of a network service that is rarely or infrequently requested by users. Basic services are usually well understood and have been used extensively over a long period of time. An extra feature that has been added for a very specific purpose or very recently is often less understood and has not been exposed to such a large number of input data (and test cases). Therefore, it is more likely that the implementation of these features contains security flaws that can be exploited. The recent, well-known attacks against the `bind` implementation of the domain name service (DNS) are not targeted against the standard name translation routines but against the code that handles NXT (next record) or TSIG (transaction signature) [4, 5] queries. When only half of all Remote-to-Local exploits abuse infrequently used features, the probability that such a request contains malicious payload is much higher than that of a regular one.

[2] Ten false alarms per day are considered to produce an acceptable low noise level that allows the system to be used in a production environment.

We therefore assign higher anomaly scores to requests that are of types that occur less frequent. The anomaly score (AS) is calculated as follows

$$AS_{type} = -\log_2 (p[typ])$$

`p[typ]` is given as the probability that a certain request is of type `typ`. This probability is equal to the relative frequency that a request with type `typ` has occurred during the training period. In order to prevent too high anomaly scores (or even infinite values) for request types that occur very infrequent (or not at all during the training period) the probability of each request type is set to be at least $3.05 * 10^{-5}$ (yielding a maximal anomaly score of 15.0).

3.2.2 Length of Request

The length of a request can be a good indicator for the correctness of its content. Usually, a service request consists of some protocol specific information and user (or user program) supplied input. The length of the protocol information does not vary much between requests of a certain type. The user supplied input mostly consists of a few, short strings (e.g. a domain name or URL) in human readable form and does not cause much variation in the total length either. The situation looks different when requests carry input to overflow a buffer in the target service. It is necessary to ship the shell-code itself (which has a typical length of a few hundred bytes) and additional padding that depends on the length of the buffer which is targeted. Instead of a short URL or a simple domain name the user supplied part contains several hundred bytes. This obviously increases the total length of the request. The anomaly score is calculated by using the mean (μ) and the standard deviation (σ) of the lengths of the requests that have been monitored during the training period. The following formula is used for a request with the length l. The anomaly score grows exponentially as the request length increases. In order to tolerate a reasonable amount of deviation of the length values, we use 1.5 as the base and multiply σ with a constant factor of 2.5.

$$AS_{len} = 1.5^{\frac{(l-\mu)}{2.5*\sigma}}$$

This formula has the property that it assigns anomaly scores greater than 1.5 (the base of the exponential function) only to requests that are longer than the average. This is consistent with our assumption that malicious payload increases the total length. The maximum value of AS_{len} is limited to 15.0.

3.2.3 Payload Distribution

The biggest advantage in extending the application model to consider the payload of requests is the possibility to analyze it for the occurrence of abnormal content. As we do not intend to do signature based analysis we have to build a model of a 'normal' payload. Our model bases on the observation that requests mainly contain printable characters and human readable strings. For example, a HTTP request consists of several plain text lines and DNS queries contain domain names as strings. This implies that a large percentage of characters in such requests are drawn from a small subset of all 256 possibilities (ASCII values for letters, numbers and a few special characters). Like in English text, those characters are not uniformly distributed but occur with different frequencies. Obviously, we cannot expect that the frequency distribution is identical to a standard text. Even the frequency of a certain character (e.g. the frequency of letter `'e'`) varies tremendously between requests. Nevertheless, there are similarities between the character frequencies in different requests. These become apparent when the relative frequencies of all possible 256 characters are sorted in descending order (obviously, many will be 0 for a typical request). Our payload analysis only bases on the frequency values themselves and it does not matter whether the character with the most occurrences is an `'e'` or a `'.'`. We call the sorted, relative character frequencies of a request its *character distribution*.

Consider the text string 'Aaaza' with the corresponding ASCII byte values '65 97 97 122 97'. The left diagram of Figure 1 shows the absolute occurrences of the bytes that are contained in the string above. The right diagram displays the sorted, relative frequencies (i.e. character distribution) which have been calculated from the absolute values (represented as a *histogram*).

For the payload of a regular request one can expect that the relative frequencies slowly decrease in value when one moves in the direction of the positive x-axis. In case of abnormal payload the frequencies can drop extremely fast (because of a peak caused by a very high frequency of a single character) or barely (in case of a nearly uniform character distribution).

The character distribution of a perfect normal packet is called *payload distribution ($\mathcal{PD}$)*. The payload distribution is a discrete distribution with

$\mathcal{PD} : \mathfrak{D} \mapsto \mathfrak{P}$ with $\mathfrak{D} = \{n \in \mathcal{N} | 0 \leq n \leq 255\}$ and $\mathfrak{P} = \{p \in \Re | 0 \leq p \leq 1\}$

The relative frequency of the character that occurs n-most often (0-most denoting the maximum) is given as $\mathcal{PD}(n)$. When the histogram in Figure 1 is interpreted as a payload distribution then $\mathcal{PD}(0) = 0.6$ and $\mathcal{PD}(1) = 0.2$.

The payload distribution is calculated during the training period. In this period the SPU stores the character distributions of all received requests. The payload distribution is then approximated as the mean of all character distributions. This is done by setting $\mathcal{PD}(n)$ to the mean of the frequencies for the n-most frequent character of all requests. As we have operated on relative frequencies that sum up to 1.0, the means will do so as well (making the payload distribution well-defined).

For each request received in detection mode, we assume that the character distribution is a sample drawn from the payload distribution. We use a statistical test to determine the likelihood that the sample is really derived from the payload distribution. For a normal request the test should yield a high confidence in the correctness of this hypothesis while it should be rejected for malicious payload. We use a variant of the Pearson χ^2-test as our 'goodness-of-fit' test.

For our intended statistical calculations it is not necessary to operate on all values of $\mathcal{PD}$ directly. Instead, it is enough to consider a small number of intervals. Therefore, we divide the domain of $\mathcal{PD}$ into a total of six segments (as shown in the table below).

Segment	0	1	2	3	4	5
x-Values	0	1-3	4-6	7-11	12-15	16-255

The expected relative frequency of characters in a segment can be easily determined by adding the values of $\mathcal{PD}$ for the corresponding x-values. As the relative frequencies are sorted in descending order we expect the values of $\mathcal{PD}(n)$ to

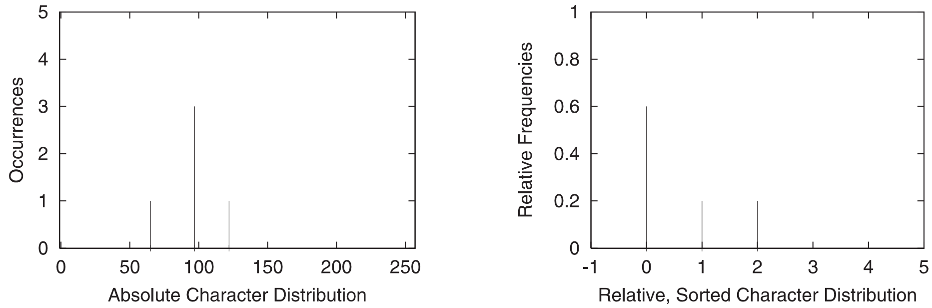


Figure 1: Character Distributions

be more significant for our anomaly score when n is small. This fact is clearly reflected in the division of $\mathcal{PD}$'s domain.

When a new request arrives the absolute number of occurrences for each character is determined. Afterwards, these values are sorted in descending order and combined according to the table above (aggregating values that belong to the same segment). The χ^2-test is utilized to calculate the probability that the given sample (derived from this request) has been drawn form the payload distribution. The standard test requires the following steps to be performed.

1. *Calculate the observed and expected frequencies* - The observed values O_i (one for each segment) are already given and the expected number of occurrences E_i are calculated by multiplying the relative frequencies for each of the six segments with the length of the request.

2. *Compute the χ^2-value* as $\chi^2 = \sum_{i=0}^{i<6} \frac{(O_i - E_i)^2}{E_i}$

3. *Determine the degrees of freedom and obtain the significance* - The degrees of freedom for the χ^2-test are identical to the number of addends in the formula above minus 1. This yields 5 in our case. The actual probability that the sample is derived from the payload distribution (i.e. significance of the sample) is read from a predefined table using the χ^2-value as index.

The χ^2-values themselves increase as the likelihood that the sample stems from the payload distribution decreases. Therefore, it is not necessary to first perform the table lookup in step 3 and then transform the probability back into an anomaly score. The χ^2-value can be used directly for the computation of the score. As stated above, the test operates on absolute values. This results in higher absolute χ^2-values for longer packets than for short ones even though they have the same relative deviation from the payload distribution. As the packet length is already factored into our anomaly score, we divide the χ^2-value by the payload length `l` to get a length independent result. This result is then multiplied by a constant factor of 15.0 (the maximum used for both other properties) to get scaled to the correct order of magnitude. The maximum is set to 20.0 (in contrast to 15.0 to reflect the higher importance of this property).

$AS_{pd} = \chi^2 * \frac{15}{l}$

This approach is surprisingly efficient. In the following Section 4 we show by means of experimental data that this method is able to distinguish between normal and malicious requests. It has the additional advantage compared to signature based systems that it cannot be fooled by some well known attempts to hide the attacker's payload. Signature based systems often contain rules that cause an alarm when long sequences of 0x90 bytes (`nop` operation of Intel x86 based architectures) are detected in a packet. As a consequence, attackers substitute such sequences with assembler instructions that act similar (e.g. `add rA, rA, 0` which adds 0 to the value in register A and stores the result back to A). This prevents signature based systems from detecting such attacks. In our case, such sequences still cause a distortion of the request's character distribution and result in high anomaly scores. In addition, characters in malicious payload are often `xor`'ed with constants or shifted by a fix value (ROT-13 code). Such evasion attempts do not change the resulting character distribution and the anomaly score remains the same.

3.3 Anomaly Score

The anomaly score is a value that specifies the extent of the deviation of the received request from the expected values specified by the profile. It is a compound value derived from the factors that have been described above and calculated as follows.

$AS = 0.3 * AS_{type} + 0.3 * AS_{len} + 0.4 * AS_{pd}$

The anomaly score for each request can be in a range from 0 to 17.00 (when each addend contributes to the sum with its maximum). The payload anomaly score has slightly more weight to reflect its importance. This score is compared to a threshold that can be chosen by the security administrator. The default threshold is computed during the training phase and set to a value that would cause 10 false alarms per day when the system would receive the training data itself as input. A lower threshold means that is more likely that attacks are detected with the disadvantage of an increasing number of false alarms. The limit should be set to the lowest value possible provided that the number of false alarms is manageable. This decision depends on the type of traffic that is seen on the network and a policy which decides how many false alarms are considered acceptable. The next section shows that our system managed to detect all our attacks by setting the threshold to a value that produced significantly less than 10 false alarms per day during our experiments.

4. EVALUATION

We have implemented a prototype that can process HTTP and DNS traffic. Because of lack of space only the results

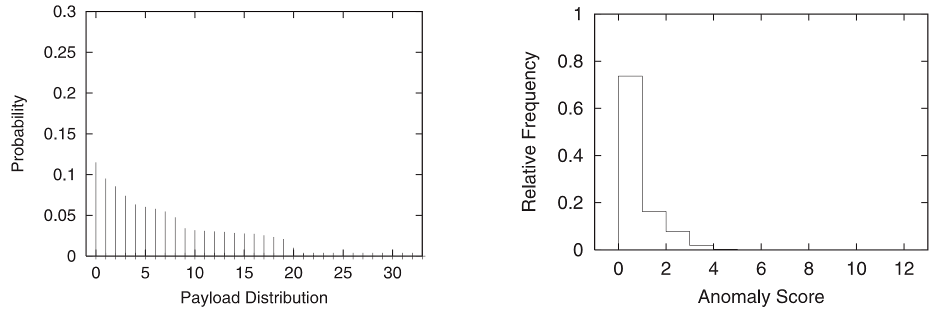


Figure 2: Payload and Anomaly Score Distribution

Score	Absolute	Relative	Score	Absolute	Relative
[0, 1[	507041	0.73653	[6, 7[	76	0.00011
[1, 2[	112319	0.16316	[7, 8[	14	0.00002
[2, 3[	53294	0.07742	[8, 9[	6	0.00001
[3, 4[	13136	0.01902	[9, 10[	0	0.00000
[4, 5[	2240	0.00325	[10, 11[	0	0.00000
[5, 6[	262	0.00038	[11, 12[	0	0.00000

Table 1: Absolute and Relative Distribution of Anomaly Scores

for one protocol can be presented in detail. We have chosen DNS because the DNS protocol uses a simple compression mechanism to shorten the length of requests by substituting substrings of domain names with pointers to previous occurrences of these strings. This could have a negative effect on our assumption regarding the character distribution. In contrast to that, HTTP requests are sent in plain text. Below, we show the results of our service specific anomaly detection system that has been installed on the DNS server of our department.

The service independent part of the packet processing unit (PPU) has been realized with Snort [15]. Snort is an open-source, signature based network intrusion detection system that has the ability to reassemble IP and TCP traffic. It offers an interface that allows developers to use custom modules as plugins. We realized the service dependent part of our PPU as such a plugin that is inserted directly after Snort's IP/TCP reassembly stage. This allows us to operate on completely reassembled UDP packets or acknowledged chunks of TCP streams.

Usually, DNS uses a single UDP packet to transmit a request to the server. In such a case, the complete payload of the packet can simply be passed to the statistical processing unit (SPU). In case of a request that is transmitted over a TCP stream, the PPU connects subsequent TCP stream chunks which it receives from the TCP reassembler. The simple header of the DNS request is parsed to determine the amount of data that is transmitted. When the complete request has been observed, it is passed to the SPU.

It is not entirely obvious how the type of a DNS request should be determined. In order to save overhead, each request contains a number of records (called resource records) that usually have different types. Nevertheless, each DNS request contains a distinguished first resource record (called question) that specifies the desired operation. We use the type of that question resource record to calculate the type dependent anomaly score.

The following two tables show the application model that was built during a training period of 24 hours. A total of 75463 requests with average length (μ) 39.572 and variance (σ) 31.915 have been processed. The following table shows the absolute and relative occurrences of requests with respect to their type.

Type	Explanation	Occurrences	relative Freq.
PTR	Reverse DNS Query	57306	0.7594
A	DNS Query	15963	0.2115
ANY	Request all Records	1167	0.0155
AAAA	IPv6 Query	599	0.0079
MX	Mail Exchange Query	317	0.0042
SOA	Zone of Authority	111	0.0015
Total		75463	1.0000

The table below shows the expected character frequencies for all six segments as determined by the payload distribution. The first 32 values of the payload distribution are displayed in the left diagram of Figure 2.

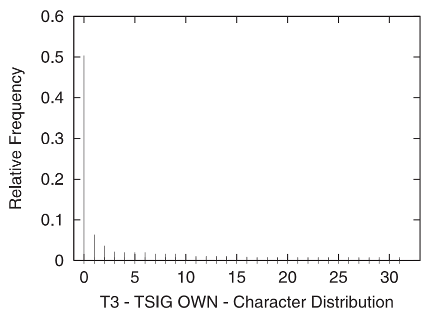
Segment	0	1	2	3	4	5
Expected Freq.	0.117	0.257	0.185	0.199	0.117	0.1253

After the initial training period we used the resulting application model to test our ID system on the department's DNS server for 10 days. During this time our system processed 688388 DNS requests. Table 1 above shows the absolute and relative number of requests with respect to their anomaly score. In addition, the relative numbers are visualized in the right diagram of Figure 2.

We have increased the default threshold of 6.97 to 7.0 and received only 20 false alarms during the complete test period (an average of only 2 per day). The false alarms have been exclusively caused by very short requests that contained less than 20 characters. As the χ^2-test is inaccurate for very small samples their A_{pd} score was the maximum possible. Additionally, they have been of types that only occur infrequently. Our system could be modified to put lesser weight on the χ^2-value for small samples to reflect this inaccuracy.

Test Case	Seg. 0	Seg. 1	Seg. 2	Seg. 3	Seg. 4	Seg. 5
IQuery	13	7	3	4	0	0
	3	7	6	5	3	3
Tsig LSD	180	105	33	35	18	138
	59	130	93	102	60	65
Tsig OWN	275	67	33	40	23	108
	64	139	101	108	65	69
Infoleak	11	8	3	2	0	0
	3	6	4	5	3	3
Tsig Lucy	121	69	39	53	36	215
	62	135	100	106	63	67
Nxt	61	39	31	46	32	348
	64	143	102	112	66	70

Table 2: Expected and Actual Character Distribution of Attacks



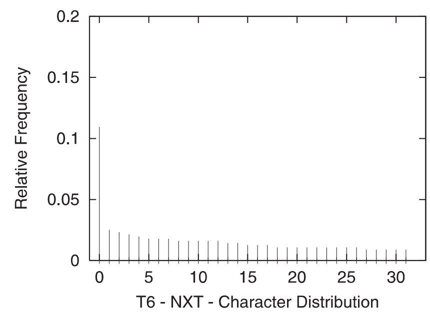


Figure 3: Exploit Character Distributions

In that case, the threshold of 7.0 might have resulted in no false alarms at all.

The payload of all requests that have been received during the test period were dumped into a file with the final size of 67.2 MB. We measured the additional load that our traffic analysis inflicted on the DNS server by running our system off-line on that dump file. The average runtime (after executing the program ten times) on a Pentium 4 (1.4 GHz - 512 MB RAM - Linux 2.4.2) to process the complete file was 41.3 seconds. As 10 days worth of data could be analyzed in under a minute the additional load on the DNS server was negligible.

After the encouraging false alarm rate has been explored we attacked our DNS server with several actual exploits (obviously we use the latest, patched version of `bind`). Our tests included five DNS exploits listed in arachNIDS [1] (a well known exploit database) and additionally the famous but now outdated NXT ('ADM rocks') exploit. The arachNIDS site lists a total of 8 offending signatures of potential attacks against port 53 (used by DNS). Three of them describe regular requests (e.g. requesting a zone transfer) that might be used by an attacker to get information. They are not included in our tests because they can be used in unmodified form by authorized clients to perform legal requests and therefore should not raise an alarm.

The following table lists properties and anomaly scores (AS) for all six test cases. All exploits use requests of types that have not occurred during our training phase and received the maximum A_{typ} scores.

Test Case	Length	A_{typ}	A_{len}	A_{pd}	AS
T1 - IQuery	27	15.0	1.07	14.48	10.61
T2 - Tsig LSD	509	15.0	10.86	8.99	11.36
T3 - Tsig OWN	546	15.0	13.11	16.36	14.98
T4 - Infoleak	24	15.0	1.08	13.74	10.32
T5 - Tsig Lucy	533	15.0	12.27	9.43	11.95
T6 - Nxt	557	15.0	13.87	20.00	16.66

Table 2 above shows for each exploit request the actual and expected character frequencies for all six segments that have been used to calculate A_{pd}. Figure 3 shows typical character distributions of two exploits. The left diagram shows a distribution with a very large slope while the distribution in the right one is too uniform.

All malicious requests have anomaly scores greater than the threshold and have been correctly identified as attacks. Notice that in addition to that all intrusions have anomaly scores that are larger than those of every regular request received during normal operation. This indicates that the threshold could be raised without missing attacks.

5. CONCLUSION

We have presented an intrusion detection system that uses statistical anomaly detection to find Remote-to-Local attacks targeted at essential network services. We use a service based approach that separates statistical data for requests to different services to improve our detection capability. In contrast to other systems which mainly rely on information in the network and transport layer headers (TCP/IP) to perform their analysis we propose an extended application model that includes the payload as well. We have described our method to obtain the actual application model and to calculate anomaly scores for service requests. The feasibility of this approach is justified by experimental data from our DNS specific intrusion detection evaluation.

6. REFERENCES

[1] arachNIDS: advanced reference archive of current heuristics for Network Intrusion Detection Systems. http://www.whitehats.com/ids, 2001.

[2] M. Bykova, S. Ostermann, and B. Tjaden. Detecting network intrusions via a statistical analysis of network packet characteristics. In *Proceedings of the 33rd Southeastern Symposium on System Theory*, 2001.

[3] J.B.D. Caberera, B. Ravichandran, and R.K. Mehra. Statistical traffic modeling for network intrusion detection. In *Proceedings. 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2000.

[4] CERT Advisory CA-1999-14 Multiple vulnerabilities in BIND. http://www.cert.org/advisories/CA-1999-14.html, 1999.

[5] CERT Advisory CA-2001-02 Multiple vulnerabilities in BIND. http://www.cert.org/advisories/CA-2001-02.html, 2001.

[6] Dorothy Denning. An intrusion-detection model. In *IEEE Symposium on Security and Privacy*, pages 118–131, Oakland, USA, 1986.

[7] Laurent Eschenauer. Imsafe. http://imsafe.sourceforge.net, 2001.

[8] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[9] A. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *USENIX Security Symposium*, 1999.

[10] MIT Lincoln Labs. DARPA Intrusion Detection Evaluation. http://www.ll.mit.edu/IST/ideval, 1998.

[11] Wenke Lee, Sal Stolfo, and Kui Mok. A data mining framework for building intrusion detection models. In *In Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 1999.

[12] Peter G. Neumann and Phillip A. Porras. Experience with emerald to date. In *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, pages 73–80, Santa Clara, California, USA, April 1999.

[13] Phillip A. Porras and Peter G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th NIS Security Conference*, October 1997.

[14] Phillip A. Porras and Alfonso Valdes. Live traffic analysis of TCP/IP gateways. In *Internet Society's Networks and Distributed Systems Security Symposium*, March 1998.

[15] Martin Roesch. Snort - lightweight intrusion detection for networks. In *USENIX Lisa 99*, 1999.

[16] Dug Song. Fragrouter. http://www.monkey.org/∼dugsong/, 2000.

[17] Stuart Staniford, James A. Hoagland, and Joseph M. , McAlerney. Practical automated detection of stealthy portscans. In *Proceedings of the IDS Workshop of the 7th Computer and Communications Security Conference*, Athens, 2000.

[18] G. Vigna and R. Kemmerer. Netstat: A network-based intrusion detection system. In *Proceedings of the 14th Annual Computer Security Applications Conference*, December 1998.