Protection Against DDoS Attacks Based On Traffic Level Measurements

Boldizsár Bencsáth István Vajda
Laboratory of Cryptography and Systems Security (CrySyS)
Department of Telecommunications
Budapest University of Technology and Economics, Hungary
{bencsath, vajda}@crysys.hu

Keywords DDoS attacks, traffic analysis, network protection

***Abstract*— A method for protecting an Internet server against a bandwidth-consuming DDoS attack is proposed and analyzed. Incoming traffic is monitored continuously and "dangerous" traffic intensity rises are detected. Such an event activates a traffic filtering rule which pushes down the incoming aggregate traffic to an acceptable level by discarding excess packets according to the measured relative traffic levels of active sources. Compared to other studies, our method has a structurally stronger base: legitimate traffic to the server is not necessarily hindered because of the attack or the traffic suppression. The method is supported by an analysis and a simulation as well.**

1. INTRODUCTION

During a Distributed Denial of Service (DDoS) attack the target is an Internet server and the attacker's aim is to exhaust the server's resources in order to prevent authorized access to services or degrade the quality of service. The attacker utilizes more attacking clients and acts organized to multiply the impact of the attack and to avoid being discovered and filtered. (see [1] for a comprehensive description of DDoS attacks).

It is a fairly easy task to deploy a DDoS attack. Various prewritten tools are available on the Internet.

Protection against DDoS attacks highly depends on the model of the network and the type of attack. Although several solutions and methods have been proposed, most of them have weaknesses and fail under certain circumstances.

Protocol reordering and *protocol enhancement* are workable methods to make security protocols more robust and less vulnerable to resource consumption attacks (a few examples are listed in [2] and [3]). This approach tries to make a protocol less vulnerable against a single source of attack, but it is not so effective against a distributed attack.

Stateless protocols eliminate problems that follow from memory overload. However it is achieved at the cost of transforming the memory overload problem into a network load problem and at the same time the chance for message-replay attacks is also increased. So, stateless protocols can be useful in specific cases, but do not provide a general solution. (A study on the use of stateless protocols against DoS can be found in [4]).

A good starting point against any type of attack could be the of the initiator identification. There are tools for *tracing the source* of an attack, for example *IP Traceback* (see [5] and [6]). Unfortunately this approach in a sense contradicts one of the values of the Internet: anonymity. For example a network using onion routing technology provides complete anonymity. Traceback algorithms to identify attackers might turn out to be completely useless on such a network. The goal is to catch the 'real' initiator, not the attacking computer, but this tracing technique can only identify the latter. During a DDoS attack the number of attacking sources can be very high; Identifying and disarming all of them is not a simple task.

Several methods, like *ingress filtering[7], rate control, distributed rate control mechanisms and pushback[8]* make the attacker's work more difficult, but at the same time these tools open up new areas for attacks. If for example MULTOPS (a router technique, see [9]) were widely deployed, we would not have enough capacity on the network elements to rely on. On the other hand, if it is not widely deployed, its functioning is very restricted.

Client side puzzle and other *pricing algorithms* (see [10], [11], [12], [13]) are effective tools to make protocols less vulnerable to depletion attacks of processing power, but in case of distributed attacks their effectiveness is an open question.

We provide a simple and robust protection algorithm based on easily accessible information at the server. This is done in such a way that the server host cannot be disabled by an attacker and as soon as the overload disappears, the normal service quality resumes automatically (e.g. without restarting the server). Simultaneously we wanted to minimize the number of legitimate sources blocked by the server as it is typically needed in a real-life scenario. This criterion is not met by many of the current commercial solutions. (see [14] for a review of some commercial solutions) Our approach does not need to modify network elements outside the victim server.

The structure of the paper is the following:

Section 2 gives the description of the proposed algorithms together with the models for the attack and the traffic. Section 3 contains simulation results with emphasis on the sensitivity of the parameters of the algorithm. A prototype application is described in Section 4. Summary is given in Section 5, finally the mathematical analysis on the error rate of the attack detection, the problem of false identification is described in the Appendix.

2. DDoS Front-end Module

2.1. Traffic Model And Attack Model

In our traffic model "packets" from the network mean small stand-alone queries to the server (e.g. a small HTTP query or an NTP question-answer). For simplicity we assume that every query causes the same workload on the server: by using the appropriate enhancements (protocol enhancements, crypto hardware, caching...) on the server the workload of the different queries can be very similar. Every query causes some workload (memory and processor consumption) and thus after a certain level, the server cannot handle the incoming traffic.

The attacker uses number A hosts during the attack. When A=1 the attack originates from a single source, while the case of $A > 1$ corresponds to a distributed attack (DDoS) (see [1]). There are one or more human attackers behind the attacking sources. These attacking sources are machines on the net controlled (taken over) by the attacker for the purpose of the attack. We assume that the attacking machines use real addresses, consequently they can establish normal two way communication with the server, like a host of any legal client. The human attacker hides well behind the attacking machines in the network, which means that after carrying out the attack and after removal of all compromising traces of attack on the occupied machines, there is no way to find a trace leading to him.

Two types of sources are distinguished: legal sources and attacking sources. There are $N(t)$ legal sources and $A(t)$ attacking sources in time slot t. In our model the attacker can reach his aim only if the level of attacking traffic is high enough compared to the level under normal operation. It is assumed, that the attacker can control the extra traffic by changing the number of attacking machines and the traffic generated by these machines. In this respect we assume a powerful attacker; he can distribute the total attacking traffic among attacking machines at his choice. We assume that the reason for using several attacking machines is to make it more difficult for the server to identify and foil them all. Note, however, when more attacking machines are used by the attacker it becomes more difficult for him to hide.

Therefore, we assume that the attacker limits the number of attacking hosts ($A(t)$ is low).

In fact, a trade-off can be identified between the ability to hide and the efficiency of the attack.

2.2. The Module

A DDoS front-end module is attached to the server from the network side. (The front-end can be a software component of the server, a special hardware in the server or an autonomous hardware equipment attached to the server)

The front-end and the server together constitute a virtual server. The incoming traffic enters a FIFO buffer[1]. Discrete time model is assumed, i.e. the time is slotted. Traffic is modelled and processed per time slot. The server empties μ storage units per time slot from the buffer. Because the buffer is fed by random traffic, there is a positive probability of the event that overflow occurs. When a DDoS attack begins, the incoming traffic quickly increases and the buffer becomes full. Most of the incoming units will be dropped, so the attacker achieves his aim of significantly degrading the service quality, essentially cutting the connection to the server. However, the server host will not be disabled and remains intact. The goal of the front-end module is to try to suppress the traffic from the attacking sources effectively. (Similar method is used in [15] against SYN attacks)

[1]In practice the buffer can be a set of httpd child processes dealing with incoming queries.

Assume that there are two states of the incoming channel: normal state when there is no DDoS attack and attack state when the server is under attack. The attack begins at time t^* (time is measured in time slots). When at time $t^* + \delta$ the front-end buffer becomes full, the transport protocols run by legal and attacking hosts sense that no (or very rare) acknowledgements arrive from the server and they get effectively stuck in repeated transmissions. The first task is to detect the beginning of an attack, and estimate time t^* as close as possible.

The next task is to identify the sources of the attack and to suppress their traffic. The identification can be based on statistical properties of the traffic flow. The front-end can identify all active sources, can measure the traffic generated by these sources and can classify them into specific sets. Note, that in order to get reliable measurements for traffic level, we have to carry out these measurements in time slots between t^* and $t^* + \delta$. Consequently, the effectiveness of such protection is strongly affected by the time duration δ, during which the traffic flow is "undistorted" between the server and the sources. Therefore we should try to lengthen time duration δ to gain more time for traffic measurements. The only way seems to be the use of a huge buffer. Assume therefore, that the buffer length is $L = L_1 + L_2$, where:
length L_1 is designed to serve the normal state, assumed to be chosen according to the service rate μ and the accepted probability of accepted loss (loss is an event when incoming units are dropped because the buffer is full),
length L_2 corresponds to the excess length, with purpose to gain enough time for measurements during the start-up phase of the attack.

For simplicity, we assume, that the system has not been attacked for a long time, and the attack begins at time t^*, which means that all attacking sources start emitting packets from this time: the network is in normal state for $t < t^*$ and turns into attacked state in time t^*. Let $\hat{t}$ denote our estimate on t^*.

Furthermore we make the simplifying assumption, that the set of active sources is constant during the attack.

$T_n(t)$ denotes the aggregate traffic from the legal sources (normal traffic) and $T_a(t)$ denotes the aggregate of attacking traffic. Let the corresponding mean values (per time slot) be denoted by λ_n and λ_a, respectively, i.e.

$$E(T_n(t)) = \lambda_n, E(T_a(t)) = \lambda_a \quad (1)$$

Similarly let the corresponding standard deviations be denoted by σ_a and σ_n. Let Q denote the a priori unknown ratio between averages λ_a and λ_n, i.e $Q = \lambda_a/\lambda_n$.

Because typically the beginning of the attack (t^*) is earlier than the time of its detection ($\hat{t}$), we waste precious time for efficient traffic measurements. To minimize this loss, we estimate the aggregate traffic level continuously using sliding window estimates. The front-end handles two sliding time windows, a longer (with

capacity of w_ℓ) and a shorter one (with capacity of w_s slots). This way we measure both a long time average level, $\bar{\lambda}(t)$ and a short time average level $\hat{\lambda}(t)$, of incoming aggregate traffic per slot at time slot t.

2.3. Algorithms Of The Module

The algorithm of protection consists of the following sub-algorithms, which are executed consecutively:

A.) detection of the attack
B.) identification of the attacking sources
C.) suppression of the attacking traffic
D.) checking of the success of the suppression

A.) Detection Of The Attack

An early detection of the attack is a vital point of the protection. (In paper [16] wavelet methods are used for DDoS attack detection). In our approach, we define three simple and robust algorithms for the detection of the DDoS attacks:

Algorithm A1.

The beginning of the attack is decided to be time $\hat{t}$, which is the time, when the following event occurs:
Event 1: The buffer length exceeds L_1

Algorithm A2.

The beginning of the attack is decided to be time $\hat{t}$, which is the time, the following event occurs:
Event 2:

$$\hat{\lambda}(\hat{t}) > (1+r)\bar{\lambda}(\hat{t}) \tag{2}$$

where r , $r > 0$ is a design parameter.

Algorithm A3.

The beginning of the attack is decided to be time $\hat{t}$, which is the time, when the earlier of the two events Event 1 and Event 2 occurs.

B.) Identification Of The Attacking Sources

We would like to suppress the aggregate traffic selectively at the input (front-end) by suppressing the attacking traffic as effectively as we can. Here arises the problem of distinguishing the traffic coming from attacking sources. The assumed distinguishing characteristic of the attacking sources is the higher mean of their traffic level. It is also assumed that the front-end device can measure the traffic characteristics of all active sources distinguishing them by their network address.

Starting at time $\hat{t}$, we measure the aggregate and the individual traffic levels (i.e. traffic per source). If we correctly identified an attack, i.e. $t^* < \hat{t} < t^* + \delta$, we can make measurements over the resting time $(t^* + \delta - \hat{t})$.

Let the output of this measurement be denoted by $\hat{\lambda}_r(t^*+\delta)$ and $\hat{\lambda}^{(i)}(t^* + \delta)$ for the aggregate level and for source i respectively.

As we cannot determine the exact traffic from the legal sources during the attack we use $\bar{\lambda}(\hat{t}-c)$ $(c > 0)$ as an estimate on the mean aggregate traffic level of legal sources in time interval $[t^*, t^* + \delta]$ and we gain an estimate

$$\hat{\lambda}_a = \hat{\lambda}_r(t^* + \delta) - \bar{\lambda}(\hat{t} - c) \tag{3}$$

on the mean aggregate traffic level of attacking sources. The set Z of active sources is decomposed into

$$Z = Z_n \cup Z_a \quad (Z_n \cap Z_a = \emptyset) \tag{4}$$

where Z_n and Z_a are the sets of legal sources and attacking sources respectively. The identification algorithm outputs a subset Z_a^* of Z. This set should correspond to Z_a as closely as possible.

The identification of attacking sources is made by the following algorithm:

Algorithm B1.

Find the maximum-sized subset Z_a^*=$\{i_1, i_2, ..., i_v\}$ of Z, which corresponds to sources with the highest measured traffic levels, such that

$$\sum_{j=1}^{v} \hat{\lambda}^{(i_j)}(t^* + \delta) \leq \hat{\lambda}_a \tag{5}$$

The base of this method in our model is the predicate described in the beginning of this section: The attacker tries to hide himself and therefore limits the number of attacking sources $(A(t))$ which is in trade-off with the volume of the attack. As a result of this trade-off the traffic volume from the attacking sources is higher than the traffic from the legitimate clients.

Algorithm B2.

Omit those sources from set Z which have been active at time $(\hat{t} - c)$, $c > 0$, and use Algorithm B1.

C.) Suppression Of The Attacking Traffic

Once we have successfully identified the attacking sources, the traffic suppression algorithm is straightforward:

Algorithm C.

Discard all incoming units with sources from the set Z_a^*.
First, we apply filter rules to discard any incoming packets from the identified sources (needs memory), and then, we have to delete any previously stored packets in the front-end buffer.

D.) Checking Of The Success Of Suppression

Algorithm D.

In case of successful intervention, by running *Algorithm C* the buffer length has to retire below length L_1 within a timeout *t_out*. If it does not occur we should discard packets from further active sources beginning with the one with the highest measured traffic level, followed by a new checking of success. These steps are repeated until the wanted decrease in the queue length is reached.

A conservative estimate on timeout t_out can be the following:

$$t\_out = d \cdot \frac{L_2}{\mu - \bar{\lambda}(\hat{t} - c)} \tag{6}$$

where d is a small positive integer (e.g. d=3).

The quality of protection is determined by the

1.) reliability of attack detection,

2.) probability of false identification of attacking and legal sources,

3.) time duration of the unavailability of the service caused by buffer overflow.

A detailed analysis can be found in the Appendix.

3. Simulation Results

The simulation consisted of the following elements shown in Figure 3.

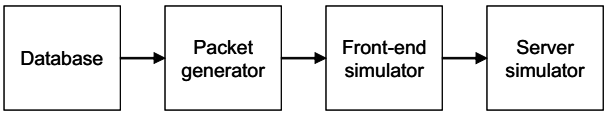

Fig. 1. The block scheme of the simulation

In addition to scheduling and buffering arrived packets, the front-end simulator collects statistical data based on the traffic, detected attacks, identified attacking sources and suppressed attacking sources.

The simulation was written in C and carried out on a PC running Linux. A MySQL database was used for storing all the data, thus our tests are fully reproducible. The packets arrived in discrete time intervals, the time resolution was set to 10^{-6} seconds. Statistics (e.g. intensity of the traffic) were gathered every second. The simulation starts at 0 second, most of the calculations are carried out only once a second. The simulation begins with 100 seconds of normal traffic to initialize the DoS front end. The attack begins after the 100-th second and lasts until the 200-th second, the simulation ends with another 100 seconds of normal traffic to gain data about the recovery of the system.

3.1. Parameter Setting

The arrivals of the various packets at the DDoS front-end are modelled by a Poisson process. These packets are stored in a buffer, and every packet is served by one virtual processor. The queue is type M/M/1, the inter-arrival times and the service times are assumed to have exponential distributions. The number of the sources in this simulation is constant in time.

The Poisson model might be judged as oversimplified for the typical Internet traffic. In this regard the aim of this first simulation effort was to see the algorithms in work and to get feedback on their basic strength and weaknesses, and the sensitivity of the parameters. At the same time, we believe that such a model might be suited to some internet protocols if the parameters are correctly set. ICMP/Ping, NTP (Network Time Protocol) and DNS clients send many small uniform-sized packets with uniformly distributed inter-send times, which makes them more similar to our simulation than other protocols (e.g. HTTP or FTP).

The number of simultaneous legitimate clients can also vary in a broad range. Some examples for different hosts can be imagined: For a small corporate server the number of legal clients is low ($N(t)$=5) the server's capacity is high, the average load is low, and thus the number of attacking hosts should be high: e.g. $A(t)$=40, so $N(t) << A(t)$. For a medium size server $N(t)$=50 and a successful attacker can deploy it's attack from a low number of hosts, e.g. $A(t)$=50, so it is easy to hide his attack ($N(t) \sim A(t)$). For a global portal there are lots of legal clients: $N(t)$=10000 and the attacker cannot easily estimate the needed number of attacking machines, so he can use $A(t)$=5000 attacking hosts with a high attacking rate (e.g. λ_a=λ_n·10) ($N(t) > A(t)$)

In our first simulation we show that our method is workable on a large number of hosts, so we decided to set the parameters as described in Table 1.

Table 1 Basic parameters of the simulation

Number of legal sources N_t	10000
Number of attackers $A(t)$	5000
λ for legal sources (λ_n)	0.1
λ for attacking sources (λ_a)	0.4
Service rate (μ) (packets/sec)	1500

In our case, described in Table 1, it is trivial that the server's capacity should be more than 1000, but the attack is only successful when the capacity μ) is below 3000 ($\lambda_a \cdot A(t) + \lambda_n \cdot N(t)$). We consider $\mu = 1500$.

The average number of packets in the buffer during normal state is

$$E(X) = \frac{(\lambda_n/\mu)}{1 - (\lambda_n/\mu)} \tag{7}$$

which gives $E(X)$=2 for data from Table 1 (see [17] for details). We set the buffer size parameter L_1=40 (packets). The other parameter that affects the length of the buffers is the overflow traffic. As the normal traffic is about 1000 packet/second, length $L_2 = 30000$ (packets) seems a safe estimate. The parameters of the algorithm for detection of attack are set as shown in Table 2.

Table 2 Attacker detection parameters

Sliding window size (w_s)	10 sec
Tolerance for traffi c jump (r)	0.6
Time frame to get last correct value of λ	45 sec

The available time for traffic measurements depends on the value of δ. This a priori unknown parameter depends on how long it takes for the buffers to fill up.

In our simulation we set a constant limit ($\hat{\delta} \leq \delta$) for traffic measurements. Suppose we know that the total traffic (with attackers) is $T_n + T_a = 3000$, and the service rate is $\mu = 1500$. We can expect the buffer (L_1) to be full after $40/(3000 - 1500) \approx 0.3$ seconds, and the whole buffer (L) after $30040/(3000-1500) \approx 20$ seconds (so $E(\delta) = 20$). It is a very safe estimation that $\delta = 10$. In real-world situations we have no preliminary knowledge about the attack and so δ (which is coherent with the L_2 buffer length) is the result of some estimation or adaptation. For simplicity, we set the short time moving average window size as $w_s = \delta$. We consider that the normal traffic is restored if the buffer length decreased under L_1.

Algorithm B1 was used during attacker identification.

3.2. Simulation 1

In this simulation our main goal was to gain data about the viability of our approach. Table 3 shows the most interesting results

of the simulation. After running the simulation with $\delta = 10$ we repeated the simulation with different window sizes. Table 3 shows clearly that a longer window size gives more accurate identification: the number of filtered legal clients decreased to 1 from 592 as the windows size increased from 5 to 40 seconds.

Table 3 First simulation results

$\hat{\delta}(\hat{\delta} = w_s)$	5	**10**	20	30	40
Correctly identified attackers	2963	**3737**	4497	4750	4875
Filtered legal clients (type II error)	601	**562**	285	148	71
Dropped packets	0	**0**	0	14361	29326
Maximum buffer level (and corresponding time frame)	29713 (200s)	**14871 (110s)**	29654 (119s)	30040 (120s)	30040 (120s)
Time to restore (after t^*)	156	**111**	79	77	83

On the other hand a large window size endangers the system with the possibility of a buffer overflow. During the simulated attack the buffer could only allow traffic measures for 20 seconds, after this the buffer filled up and after this the identification algorithm produced less accurate results.

The numbers in the maximum buffer level row show that when the time window is too short, our algorithms cannot determine the correct level of the attack. Therefore a too low number of hosts were filtered at $w_s = 5$ and this way the maximum buffer level reached a very high level at a later time. The simulation results show that our method can successfully protect the system with a good estimation of the parameters, and when enough buffers are available for measuring the traffic levels.

3.3. Simulation 2

During the second simulation we simulated a smaller system: The sample system consists of $N = 50$ legal and $A = 50$ attacker clients. Using $\lambda_n = 0.1$ and $\lambda_a = 0.2$ the task of the identification algorithm is hardened. The service rate $\mu = 8$ while the buffer lengths are $L_1 = 40$ and $L_2 = 160$. The window size parameter was considered as $w_s = \delta = 10$. Other parameters and used algorithms are the same. During Simulation 2 we repeated our tests on 500 different sets of input data to gain statistical information about the properties.

After running the simulation 500 times on different data sets we found that the attack was firstly detected by Algorithm A1 in 4 cases, Algorithm A2 was faster in 454 cases while the attack has been detected by both Algorithm A1 and A2 in 42 cases. The simulation results are summarized in Table 4.

Table 4 Simulation results

	minimum	average	confidence interval (95%)
traffic restoration time (after t^*)	51	116.624	1.8671
packets dropped	0	0.708	0.312
normal user filtered (error type II)	1	7.228	0.223
number of attackers filtered	24	34.102	0.286
attack detection time (after t^*)	0	3.07	0.09

The mean time of attack detection is $E(\hat{t} - t^*) = 3$ seconds. After this time the DDoS front-end can start to suppress the attacking traffic. The minimum detection time is $(\hat{t} - t^*)_{min} = 0$ seconds, the maximum is $(\hat{t} - t^*)_{max} = 6$ seconds.

3.4. Selection Of Parameter r

We carried out investigations on how the number of successful detections by different algorithms depends on the value of r (i.e. tolerance for traffic jump).

Table 5 The effect of parameter r on the first detection

r	Alg. A1	Alg. A2	Alg. A1+A2 same time	A1 error type II	A2 error type II
0.2	0	500	0	0	1117
0.3	0	499	1	0	234
0.4	0	495	5	0	37
0.5	0	485	15	1	9
0.6	4	454	42	0	1
0.7	18	375	107	0	2
0.8	54	273	173	0	0
0.9	138	176	186	0	1
1.0	227	106	167	1	0

The first 3 columns of Table 5 show which algorithm detected the attack firstly. Other columns show Type II error events in case of *Algorithm A1* and *A2*. The length of L_1 was set to fit to the normal traffic, so the number of type II errors by *Algorithm A1* are very low. In the range of $r \leq 0.4$ the number of false detections in case of (*Algorithm A2*) is too high.

The high number of attack detections by *Algorithm A2* in this area shows that the algorithm detects the attack correctly, but signals the attack in a too early stage. In this stage the success checking algorithm (*Alg. D*) might report the end of the attack because of the low number of packets in the buffers. Due to the lack of attacker identifications the attack detection algorithm detects the attack again. Meanwhile the traffic measurements do not reflect the correct values, because the timeframe measures contain more data from the (not fully detected) attack stage too. This makes our solution ineffective. The optimal detection would detect every attack once and identify all the attacking sources.

When r is between 0.5 and 0.9 then we can see that the number of attack detection by *Algorithm A2* decreases and for *Algorithm A1* slowly rises. For *Algorithm A2* high values of r cause that we are not detecting the attack by a traffic jump, but rather by a buffer overrun.

4. Prototype Application

To support our method we are currently engineering a prototype to apply the method on a real-world problem. Our candidate is the SMTP system with real-time virus scanning. The simple system consists of an exim MTA, an Amavis scanning engine, and a virus scanner. This system is vulnerable to DoS attacks, as the scanning and processing of an email can take 0.1-0.5 seconds and thus a sophisticated attacker against a system with a bandwidth more than 256 kbps can initiate an attack against this system. The attacker can send legitimate e-mails to one or more known recipients, so it is not obvious that the system is being attacked. This attack can also be the result of an Internet virus (e.g. sobig). Our sample scenario is shown in Figure 4.

The basic SMTP system is extended with a general TCP wrapper, a DoS front-end client and a DoS front-end engine. The

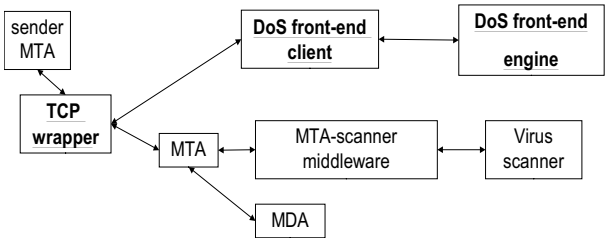

Fig. 2. Prototype topology

TCP wrapper is a simple wrapper that first asks the DoS front-end if the sender is permitted to send e-mails. After that it simply forwards the data through the connection towards the mail transport agent (MTA). The DoS front-end client is a simple application, its duty is to communicate with the DoS front-end engine and depending on the decision of the engine, send the result back to the wrapper application. The DoS front-end engine consists of a statistical engine that measures the traffic and maintains the state-variables of the DoS front-end and a decision engine that decides if a sender should be filtered out preventing to send any e-mails. If our system considers a state as an attack state, it simply sends back a regular SMTP temporary error to the sender. If the system decision is a false detection (type II error), then the sender is not able to send the message immediately, but due to the way SMTP works, after some time-out the senders SMTP server will try to send the message again. This way we can set an efficient protection against the attackers without harming the regular traffic (as we do not filter all the traffic only the attacking traffic), and we do not cause intolerable problems even if the decision is a false-positive (type II error). Our prototype is currently in the programming phase, we plan present the results of the prototype system at the conference.

5. SUMMARY

In the paper we introduced a model to handle some special cases of DDoS attacks at the victim server. We identified a trade-off at the attacker which makes the protection by traffic analysis possible. We described a simplified, but applicable model on this problem, and described all the methods and algorithms needed to successfully deploy such a protection. Our approach does not need to modify network elements outside the victim server and minimizes the number of legitimate sources blocked by the server. We gave upper bound to the error rate of the detection algorithm, and to the probability of false identification. The simulations confirm our analitical results and help in investigation of the sensitivity of the parameters

Appendix. Analysis Of The Algorithms

Reliability Of Attack Detection

In time slot t we test the following hypothesis about the state of the system

H_0: state of no attack

H_1: state of attack

There are two types of error: Type I error is the event of a missed detection, when our server is under attack and we failed to detect it, while Type II error is the event of a false detection, when an attack is detected and there is no attack.

Consequently Type I and Type II errors are the following:

$$P_I = P(\{\textit{decision on } H_0 \textit{ at time } t\}|H_1)(=0), \quad (8)$$

$$P_{II} = P(\{\textit{decision on } H_1 \textit{ at time } t\}|H_0). \quad (9)$$

We assumed that the system has been in normal state for a long time, when a change of state occurs. Considering detection algorithm A3 we can give the following rough upper bound on P_{II}

$$P_{II,A3} = P(\{\textit{queue length} \geq L_1\} \cup \{\hat{\lambda}(\hat{t}) > (1+r)\cdot\bar{\lambda}(\lambda t)\}|H_0) \quad (10)$$

$$\leq 2\cdot max\{P_{II,A1}, P_{II,A2}\} \quad (11)$$

where the individual probabilities for the Algorithms A1 and A2 are the following:

$$P_{II,A1} = P(\{\textit{queue length} \geq L_1\}|H_0) \quad (12)$$

$$P_{II,A2} = P(\{\hat{\lambda}(\hat{t}) > (1+r)\cdot\bar{\lambda}(\hat{t})\}|H_0) \quad (13)$$

Probability $P_{II,A1}$ can be calculated (designed) using approaches of standard buffer design (any corresponding textbook covers the needed techniques, see e.g. [17]). Probability $P_{II,A2}$ is considered below, where we give upper bound on it.

Theorem

Assume that random variables $T_n(t), t$=1,2,..., describing the aggregate traffic of legal sources are pairwise uncorrelated. The probability that the short time window measurement with window length w_s results in a value exceeding $r$times the mean traffic of normal state, assumed H_0 is true, can be upper bounded by

$$P\{\hat{\lambda}(\hat{t}) > (1+r)\cdot\bar{\lambda}(\lambda t)\} \leq \frac{1}{w_s r^2}\left(\frac{\sigma_n}{\lambda_n}\right)^2 \quad (14)$$

Proof: Let

$$\eta = \frac{1}{w_s}\sum_{m=1}^{w_s}\xi(m) \quad (15)$$

where

$$\xi(m) = T_n(t_m), m = 1, ..., w_s \quad (16)$$

are nonnegative, pairwise uncorrelated random variables. Applying Chebysev's inequality:

$$P[\eta > (1+r)E(\eta)] = P[\eta - E(\eta) > rE(\eta)] \leq \leq P[|\eta - E(\eta)| > rE(\eta)] \leq \frac{1}{r^2}\left(\frac{\sigma_\eta}{E(\eta)}\right)^2 \quad (17)$$

where

$$E(\eta) = E(\xi(m)) = \lambda_n\sigma_\eta^2 = \frac{1}{w_s}\sigma_\xi^2. \quad (18)$$

Bound (10) decreases with the inverse of the window length, w_s. In the case of Poisson distribution, the probability of the event that the short time window average exceeds the double of the normal traffic (i.e. $r = 1$) is upper bounded by $1/(w_s\lambda_n)$ ($(\sigma_n)^2 = \lambda_n$). ♣

A stronger upper bound can be found by using the Hoeffding bounding technique [18] if we assume the stronger condition of independence of the aggregate traffic of legal sources. In this case the corresponding bound decreases exponentially with the window length, w_s. Assume that the peak aggregate traffic level in the

normal state can be bounded by a constant K. The probability that the short time window measurement results in a value exceeding r times the mean traffic of normal state, assumed H_0 is true, can be upper bounded by

$$2 \cdot \exp\left(-2w_s\left(\frac{(1+r)\lambda_n}{K}\right)^2\right) \tag{19}$$

The Probability Of False Identification And The Problem Of Hiding

The identification algorithm outputs a subset Z_a^* of set Z of all active sources. The case when $Z_a^* \subset Z_a$ means the attacking sources are not fully identified.

The attacker wishes to hide as effectively as he can. Assume the attacker deploys the same aggregate attacking traffic independently of the number of attacking sources, $A(t)$, i.e. the use of more attacking machines means the generation of less traffic per machine. The attack is more severe when the total attacking traffic is distributed among more attacking sources. There are two extreme cases in the number of attacking sources.

Consider first the case of $A = 1$. Identification error occurs if event:

$$\max_{i \in Z_n} \hat{\lambda}_n^i > \hat{\lambda}_a^j \tag{20}$$

where $Z_a = \{j\}$ (i.e. there will be at least one misidentified source).

Let $F_n^{(i)}(k)$, and $F_a(k)$, $k = 0, 1, 2, ...$ denote the probability distribution functions of the traffic corresponding to the legal source with index i and the attacking source, respectively. The probability of event 20 is

$$P(event(20)) = \sum_{k=0}^{\infty}\left(1 - \prod_{i=1}^{|Z_n|} F_n^{(i)}(k)\right)\left(F_a(k) - F_a(k-1)\right) \tag{21}$$

where $F_a(-1) = 0$.

The other extreme case is when the number of attackers and legal users are equal and all active sources emit same traffic level. In this case, a legal user will be identified attacking with the same chance as an attacking source. The decision is cannot better as a random selection from set Z. For instance, the probability that there will be no misidentified source is practically zero.

Therefore the best strategy of the attacker is to spread uniformly the attacking traffic among as many sources as he can. If the attacker has a good estimate on parameters λ_n , $\lambda_n^{(i)}$ and μ, i.e. the aggregate level and the per source traffic level of legal users and the service rate of the server, then the most powerful attack deploys A attacking sources where

$$A = \frac{\mu - \hat{\lambda}_n}{\min\limits_i \hat{\lambda}_n^{(i)}} \tag{22}$$

Time Duration Of Unavailability Of The Service

If the identification of attacking sources is perfect, the suppression of attacking traffic frees the buffer from overload at once because all packets from attacking sources are discarded.

Therefore the time of unavailability of the service is determined by the eventual reiteration of the identification and suppression step. This complex mechanism does not seems tractable for an analysis, it is better suited for simulation.

REFERENCES

[1] Lau, F. and S. H. Rubin and M. H. Smith and L. Trajovic. "Distributed Denial of Service Attacks." IEEE International Conference on Systems, Man, and Cybernetics, Nashville, USA, 2000.

[2] Matsuura, K. and H. Imai. "Protection Of Authenticated Key-Agreement Protocol Against A Denial-of-Service Attack." In Proceedings of 1998 International Symposium on Information Theory and Its Applications (ISITA'98), pp. 466-470, Oct. 1998.

[3] Leiwo, J. and T. Aura, P. Nikander. "Towards network denial of service resistant protocols." In Proceedins of IFIP SEC 2000, Beijing, China, pp. 301-310, August 2000.

[4] Aura, T. and P. Nikander. "Stateless Connections." In ICICS'97, LNCS 1334. Springer-Verlag, pp. 87-97, 1997.

[5] Park, K. and H. Lee. "On The Effectiveness Of Probabilistic Packet Marking For IP Traceback Under Denial Of Service Attack." Tech. Rep. CSD-00-013, Department of Computer Sciences, Purdue University, June 2000.

[6] Eronen, P. "Denial Of Service In Public Key Protocols." Paper presented in Helsinki University of Technology's Seminar on Network Security course (Fall 2000), December 2000.

[7] Ferguson, P. and D. Senie. "Network Ingress Filtering: Defeating Denial Of Service Attacks Which Employ IP Source Address Spoofi ng." RFC 2827, May 2000.

[8] Ioannidis, J. and S. M. Bellovin. "Implementing Pushback: Router-based Defense Against DDoS Attacks." In Proceedings of Network and Distributed System Security Symposium, Reston, VA, USA, Feb. 2002, The Internet Society.

[9] Gil, T.M. "MULTOPS: A Data Structure For Denial-of-service Attack Detection." Vrije Universiteit, 2000.

[10] Dwork, C. and M. Naor. "Pricing Via Processing Or Combatting Junk Mail." In Advances in Cryptology. In Proceedings of the Crypto '92: 12th Annual International Cryptology Conference, Lecture Notes in Computer Science volume 740, pp 139-147, Santa Barbara, California, August 1992. Springer.

[11] Jakobsson, M. and A. Juels. "Proofs Of Work And Bread Pudding Protocols." In Proceedings of the IFIP TC7 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99), pp. 258-272, Leuven, Belgium, Spetember 1999. Kluwer.

[12] Juels, A. and J. Brainard. "Client puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks." In Proceedings of the 1999 Network and Distributed System Security Symposium (NDSS '99), pp 151-165, San Diego, California, February 1999.

[13] B. Bencsáth, L. Buttyán, I. Vajda, "A Game Based Analysis Of The Client Puzzle Approach To Defend Against DoS Attacks", Proceedings of SoftCOM 2003 11. International conference on software, telecommunications and computer networks, pp. 763-767, University of Split, 2003,

[14] Forristal, J. "Fireproofi ng Against DoS Attacks." http://www.networkcomputing.com/1225/1225f3.html, Network Computing

[15] Mutaf, P. "Defending against a Denial-of-Service Attack on TCP." In Proceedings of the Recent Advances in Intrusion Detection Conference, 1999.

[16] Ramanathan, A. "WADeS: A Tool For Distributed Denial Of Service Attack Detection." Thesis at Texas A&M University, August 2002.

[17] Gross, D. and C. M. Harris. *Fundamentals of Queueing Theory*, Wiley-Interscience; ISBN 0471170836

[18] Hoeffding, W. "Probability Inequalities For Sums of Bounded Random Variables". American Statistical Association Journal, pp 13-30, 1963.