Low Rate DoS Attack to Monoprocess Servers

Gabriel Maciá-Fernández, Jesús E. Díaz-Verdejo, and Pedro García-Teodoro

Dpt. of Signal Theory, Telematics and Communications - University of Granada,
c/ Daniel Saucedo Aranda, s/n - 18071 - Granada, Spain
{gmacia, jedv, pgteodor}@ugr.es

Abstract. In this work[1], we present a vulnerability in monoprocess or monothreaded servers that allows the execution of DoS attacks with the peculiarity that they are generated by low rate traffic. This feature makes the attack less vulnerable to detection by current IDS systems, which usually expect high rate traffic. The intruder can take advantage of some knowledge about the inter-output times in the server to build the attack. We have simulated and tested it in a real environment, obtaining worrying conclusions due to the efficiency achieved by the attack, with low effort from the attacker.

1 Introduction

Denial of Service (DoS) attacks are a very serious problem in Internet, with a great impact in the service offered by small as well as by big and important companies like Ebay, Amazon or Buy.com [1]. Besides, this kind of attacks could be carried out in a wide variety of manners, as it has been pointed out in the computer network literature [2]. This kind of attacks tries to exhaust some resources in a single machine or in a network with the aim of either reducing or subverting the service provided by them. Considering the three main aspects of security, that is, confidentiality, integrity and availability, DoS attacks focus their target in the reduction of the last one. The intruders usually achieve their goal by sending to a victim, either in a single or in a distributed way (DDoS) [2], a stream of packets that exhaust its network bandwidth or connectivity, or somehow exploiting any discovered vulnerability, causing all the cases a denial in the access to the regular clients. In recent years, there have been some large-scale attacks that affected important Internet sites [3] [4] [5], what demonstrate the vulnerability of the network environments and services to this kind of attacks.

Close to the evolution of the DoS attacks, many proposals have also appeared for preventing and detecting them. Many of the preventive measures are applicable to mitigate DoS attacks, like egress or ingress filtering [6] [7], disabling of unused services [8], change of IP address, disabling of IP broadcasts, load balancing, or honeypots [9]. However, although prevention approaches offer increased security, they can never completely remove the threat so that the systems are always vulnerable to new attacks.

[1] This work has been partially supported by the Spanish Government through MYCT (Project TSI2005-08145-C02-02, FEDER funds 70%).

J.A. Clark et al. (Eds.): SPC 2006, LNCS 3934, pp. 43–57, 2006.
© Springer-Verlag Berlin Heidelberg 2006

That is why it is necessary the adoption of intrusion detection systems (IDS) capable for detection of attacks [10]. Many efforts have been made to solve the problem of discovering DoS attacks in a high-bandwidth aggregated traffic, as it is stated in the works of Talpade et al. [11], Cabrera et al. [12] or Mirkovic et al. [13]. Besides the signature detection IDS's, the major part of the proposed IDS's for DoS detection relies on the identification of the attack by using techniques that are based on the hypothesis that it is going to be carried out through a high rate flooding from the intruder.

In this paper we analyze an exploit in monoprocess or monothreaded servers. We postulate the use of these kind of servers in simple device services in pervasive computing, so this exploit allows an intruder to carry out a kind of DoS attack, classified as an application level attack according to [14], that presents the special feature of using a low rate traffic against the server. Due to this fact, the attack will be capable of bypassing the detection mechanisms that usually rely on a high-bandwidth traffic analysis. Kuzmanovic et al. already presented in [15] an attack with similar rate characteristics and, afterwards, some solutions appeared to tackle this problem [16], [17], [18]. Although the attack presented in this work is similar to the previous ones in some aspects, there are important differences with them. Both type of attacks take advantage of a vulnerability caused by the knowledge of a specific time value in the functioning of the protocol or application, thus allowing an ON/OFF waveform attack that results in low rate traffic but with high efficiency in denying the target service. However, the attack presented in [15] is TCP-targeted, while the one introduced here threatens the application level. While the other attack generates outages in a link, this one only overflows a service running in a server, so that it could be unnoticeable in a link overflow monitoring. Moreover, in our case, the waveform of the attack is changeable along the time and the burst period does not involve a very high rate, so that the previously presented detection techniques require to be adapted. There are also differences in the vulnerabilities exploited in both kind of attacks. In the TCP-targeted (low rate) case, the knowledge of the RTO timer for the congestion control implemented in TCP is exploited, whilst in the monothreaded server case the inter-output times are the key to build the attack, as it will be presented in Section 3. But the main difference between the two attacks lies in the fact that during the TCP-targeted attack, the link is just busy in the outages periods, while in the ours one the server is always busy and processing requests from the intruder, causing the legal users the sensation that the server is not reachable. This last feature is similar to the behavior of the `Naptha` attack [19], although the main difference is that `Naptha` is a brute-force attack executed with high rate traffic.

This article is structured as follows. In Section 2 the scenario of the attack and the model of the system that is going to be attacked are defined. Next, in Section 3 it is specified how the attack is going to be performed. Section 4 will evaluate the effectiveness and behavior of the attack by means of simulations. Following, Section 5 will validate the results in a real environment, showing the goodness of the simulated results. Finally, some conclusions are presented in Section 6.

2 Modelling and Scenario Definition

We shall analyze an exploit in monoprocess or monothreaded servers that would potentially allows an intruder to carry out denial of service attacks at the application level characterized by low arrival rates at the server. The scenario to be considered is a generic client-server scenario where the server is going to receive aggregated traffic from legal users and intruders. Without lost of generality, the intruder could make the decision of carrying out the attack in a distributed manner or not, depending on the preventive measures [14] implemented in the target system and the own capabilities of the attacker.

2.1 Server Modelling

The first step in this work is to define a model that captures the behavior of a monoprocess or monothreaded server. The server will be simply a black box that receives packets, processes them and gives responses. This way, the overall system receives input messages requesting some specific service offered by a server. The system reacts to these requests by sending the message carrying the desired data (OK message in advance), or simply rising an event or sending a message indicating an overflow in the internal buffers or queues (MO event in advance). It is important to point out that, in real environments and depending on features of the considered server, this last event could be observable either through a specific protocol message, or logs, process signals, or even to be not observable at all.

The model consists of two elements: a) a *service queue* and b) a *service module*. These elements interact as shown in Fig. 1, where the arrows show the processing path followed by the incoming packets. The first building block is the service queue, where the incoming requests are temporarily stored. It could represent a TCP connections buffer, UDP buffer or simply internal application buffers. The processing of the client requests is made by the service module, which represents a service carried out by a single processing thread running on an operating system. That is, this module serves only one request from the service queue each time (iterative server).

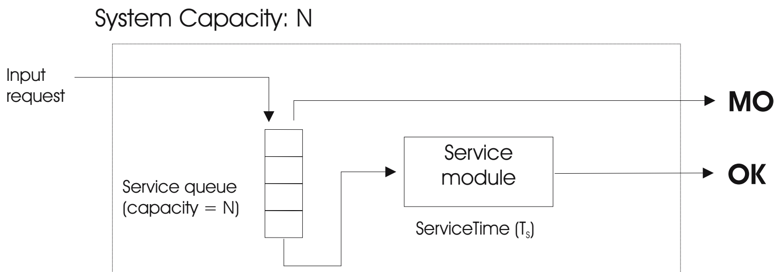

Fig. 1. Model for monoprocess/monothreaded server

The system operates as follows. An input request enters the system. If the service queue has free positions, the request is queued on it. Otherwise, an MO event occurs. The request will stay in the service queue during t_q seconds waiting for its turn to be served. This time is the so called *queue time* in queuing theory. Then, it will be processed by the service module during t_s seconds. This time is called *service time* and it corresponds to a parsing of the data, or simply the time elapsed in a complex calculation. Finally, when the processing is completed, the corresponding response to the input request is generated (OK message).

We can consider that t_q and t_s are samples from respective random processes represented by random variables. As a notation, in the following we will use lower-case letters for instantaneous values of a random variable, and their correspondent capital letters represent the general process. Moreover, operators $E[\cdot]$ and $Var[\cdot]$ will be, respectively, the mean value and the variance of the specified random variable. The variable T_s will be modelled as a normal distribution ($\mathcal{N}$).

The arrival of the user requests is also considered a random process. As recommended in many teletraffic studies [20], the arrivals are modelled by a Poisson process. Therefore, the distribution of user inter-arrival times, T_a, corresponds to an exponential probability:

$$P(T_a = t) = \lambda \cdot e^{-\lambda t} \tag{1}$$

where λ represents the arrival rate of requests from the users. Moreover, it is demonstrated [21] that the aggregation of traffic from users whose inter-arrival times are exponentially distributed, with rates λ_i, results in a traffic with inter-arrival time that also follows an exponential distribution with inter-arrival time:

$$\lambda = \sum_{i=1}^{n} \lambda_i \tag{2}$$

This expression implies that it is possible to suppose only one user in the scenario of study to represent all the aggregated traffics (Poisson processes) coming from n users.

2.2 Inter-output Time

The potential exploit that allows an intruder to carry out an attack is the knowledge of the inter-output time of the server. This parameter, named τ, is defined as the time elapsed between two consecutive outputs or answers given by the server. As stated in Section 2.1, an OK message is here termed an output or answer. On the contrary, MO messages are not going to be considered as outputs.

For simplicity, we examine a case of study in which fixed times are considered instead of random processes. Thus, the service time will be represented by the mean value of its random variable. Although this could seem a very restrictive case, it will be shown in the next section that the results are still valid for more complex cases in which the service time is not fixed.

With the above restrictions and assuming that the service queue is always full of service requests, Fig. 2 shows an example of the inter-output time diagram

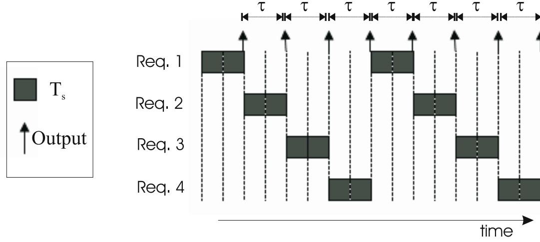

Fig. 2. Inter-output time diagram and processing state for several requests

and processing state for several requests. For a server with a service queue of 4 positions, the service time is drawn in dark grey colour. Outputs are signaled by vertical arrows. A simple observation of the scheme leads us to the conclusion that, upon the condition of fixed service time, it is obtained a fixed inter-output time τ that corresponds with:

$$\tau = E[T_s] \tag{3}$$

In the case that the service queue is not always full of service requests, the inter-output time could experience some variations, specifically when there are no requests to be served. However, the attacker can easily maintain an appropriate level of requests in the queue in order to get a fixed inter-output time.

The low rate DoS attack to monoprocess servers relies on the knowledge of this time. Nevertheless, it is necessary to indicate that this time could be still estimated by the intruder in real conditions where there are variable service times and propagation delays. In the case that variable service times and propagation delays appear, it is important to differentiate between the inter-output time from the server and that one observed by the users (legal or not). In effect, an observer of the inter-output time located at the server will appreciate a normal distribution behaviour with a mean $E[T_s]$ and a variance $Var[T_s]$:

$$\tau_{server} = \mathcal{N}\big(E[T_s], Var[T_s]\big) \tag{4}$$

However, any client of the service will appreciate a difference in the inter-output time, due to the influence of the variance in the round trip time (RTT). This way, assuming independency between the variables T_s and RTT, the observation of the inter-output time will be, in this case:

$$\tau_{user} = \mathcal{N}\big(E[T_s], Var[T_s] + Var[RTT]\big) \tag{5}$$

In what follows, we will try to validate this behaviour by means of simulation.

2.3 Inter-output Time Validation

We have used Network Simulator (NS2) [22] to check whether the assumption that inter-output time approximates to Eq. (5) is correct under non empty queue conditions. For this purpose, a new class derived from the application class that

behaves like a server (*server class*) has been implemented, which makes traces and logs to track all the events and statistics necessary for our study.

We have used two set of experiments to validate the expected results about the inter-output time of the model. The first set of experiments tries to validate the behaviour explained in Section 2.2, when fixed values for T_s are used. According to the results obtained in this case, it can be concluded that the behaviour of the simulated system is as expected.

In the second set of simulations, we have considered the case when the service time T_s and the round trip time RTT are modelled with normal distributions. The obtained results are very promising, showing low variations, in a statistical sense, among the predicted and the simulated behaviour.

Fig. 3 shows some partial results derived from the second set of experiments. In this case, the service queue has been offered user traffic with $E[T_a] = 1.0$ s. Other values used in this simulation are the capacity of the queues $N = 40$, and $T_s = \mathcal{N}(1.5\,s, 0.02)$. The round trip time takes the value $RTT = \mathcal{N}(0.6\,s, 0.02)$. It should be notice that $E[T_a]$ has been adjusted to maintain the service queue completely full, due to the fact that the user traffic rate is higher than the service rate in the server. The simulation for a period of 1000 s provides 694 outputs. The mean value obtained for inter-output time is 1.520 s, with variance 0.041, which accurately approximates the values given by Eq. (5). It is noticeable that the obtained histogram for the inter-output times approximates to the normal probability function (see Fig. 3(b)).

Fig. 4 shows other results belonging also to the second set of experiments, where there is no congestion in the service queue. In this simulation case we have adjusted the same parameters than in the previous one, but changing the user inter-arrival mean time $E[T_a]$ to 1.5 s. The same value for $E[T_s]$ have been chosen in order to maintain requests in the service queue. The simulation for a period of 1400 seconds has provided 923 outputs. The mean value obtained is 1.536 s and the variance 0.114. The deviation is greater than in congestion conditions due to the values generated when the buffer has no requests. Fig. 4 represents the inter-output times in the main axis and the occupation of the buffer in the secondary axis (dashed lines). We can see that the periods with void service queue occupation results in sporadic higher values for the inter-output times, as stated in Section 2.2.

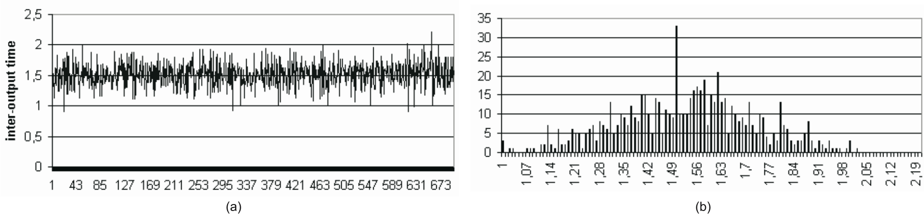

Fig. 3. Simulation of inter-output time with flooded buffer: (a) Inter-output time values and (b) histogram of the samples

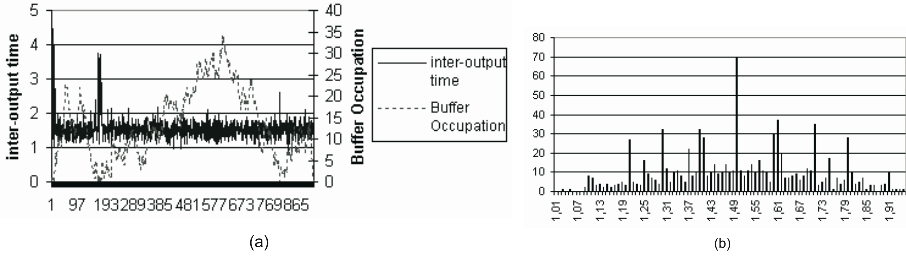

Fig. 4. Simulation of inter-output time with no flooded buffer: (a) Inter-output time values and buffer occupation level and (b) histogram of the samples

The results of this set of experiments show that the assumptions made for the operation under congestion in the service queue are good enough. It is also interesting to check that, under no complete saturation conditions, the results are still valid although they present less accuracy.

These experiments show that, even in simulated scenarios where service time is variable, the inter-output time could be still predictable for a possible intruder to build an attack based on this knowledge.

3 Low Rate Attack Specification

With the aim of demonstrating that monoprocess applications could suffer a low rate denial of service attack, we are going to specify how the attack would be accomplished.

In order to successfully attack the server, first of all, it is necessary to determine the weak point to be exploited. The resource to be exhausted is the service queue, in which the requests are queued by the application. Therefore, the objective is to maintain it full (or at least so much full as possible) of requests from the intruder. This way, legal users are not going to be able to queue their requests, thus perceiving a denial of the service provided by the application.

This strategy is commonly used in DoS attacks. However, the particular characteristic of the introduced attack here is that the intruder is not going to flood the buffer with a high rate of requests (brute-force attack), but a low rate traffic. The way to achieve this goal is by making a prediction of the instants at which the server responds to the requests, that is, when an output is going to be generated. The intruder will flood the service queue only during a short period of time around the predicted output time, resulting in an overall low rate flood experienced by the target server.

The low rate attack will have two phases: a *transitory phase*, detailed in Section 3.2, which will pursuit to initially flood the service queue, and a *permanent phase*, explained in Section 3.1, through which we try to maintain it full (or at least so much full as possible) of requests by sending request packets at a low rate.

3.1 Permanent Phase Execution

The aim of the permanent phase of the attack is to maximize the time during which the service queue is full of requests. This implies that every released position in the queue has to be seized by an intruder request in the minimum time. When the system is full of requests, a free position is issued when the server provides an output or answer. It is therefore necessary to concentrate the intruder efforts in the moment in which the server raises it, in order to seize the new free position in the service queue. Therefore, the attacker should, ideally, send the requests in such a way that they arrives at the server synchronized with the outputs.

We concluded in Section 2.2 that the perceived time between two consecutive outputs is a random process represented by τ_{user}. However, the statistical nature of the process will introduce some variability in the instantaneous values of τ_{user}, which should be considered in the attack strategy. It is also important to note that the inter-output time is independent of the queue time, and only the service and round trip times determine it. Besides, at the time we are trying to seize positions in the service queue, there will be other users that are attempting to succeed on this same purpose. So, in order to raise the probability of filling the free position with low rate traffic when an output happens, the intruder should synchronize the sending of the requests with the output in the server. To do this, the intruder use a ON/OFF scheme to flood the server with the ON phase situated in the moment of the output.

The proposed attack strategy consists in consecutive periods composed by a period of activity, *ontime*, followed by a period of inactivity, *offtime*. The attack is specified by the following parameters, as depicted in Fig. 5:

- *Estimated mean inter-output time* ($\overline{E[\tau_{user}]}$): it is an estimation of τ_{user} made by the intruder. This can be easily obtained by sending close requests and evaluating the time between the corresponding responses. Of course, the answers should be consecutive in the server. After some proofs, the intruder will succeed on this aim.
- *Offtime:* time during which there is no transmission of attack packets.
- *Ontime*: time during which an attempt to flood the service queue is made by emitting request packets.

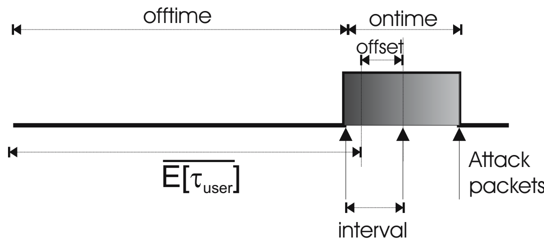

Fig. 5. Attack parameters

– *Interval*: period of time elapsed between the sending of two consecutive packets during *ontime*.
– *Offset:* represents the deviation between the center of *ontime* and $\overline{E[\tau_{user}]}$. The value 0 for the *offset* indicates that *ontime* is completely centered in $\overline{E[\tau_{user}]}$.

Both *offtime* and *ontime* must be adjusted in such a way that a synchronization between the output and the reception of the requests from the intruder is achieved. Thus, *ontime* should be centered around the estimated mean inter-output time and its duration should be proportional to twice its variance to account for its variability. The *offset* is considered to shift the whole *ontime* period in order to take into account second order aspects, as those related with discrete timing in the attack packets during *ontime*.

Although the main strategy of the permanent phase has been outlined, there are some important details to be considered. First, the effect of the statistical variability of T_s and RTT should be taking into account. In order to avoid the cumulative sum of variations through several periods, the intruder will reestimate on each period the expected timing for the next output. So, the attack period (*offtime / ontime*) will be restarted each time a response is received from the server. Obviously, the reception of an output during the *ontime* period implies to stop the current period start a new one. Besides, it is important to take into account the effect of the round trip time between the intruder and the victim. To consider this, the initially calculated *offtime* value should be reduced to *offtime*$-E[RTT]$.

Fig. 6 shows a diagram of the execution of the permanent phase that can be used to review the different steps of the attack. First, the *ontime* period starts by sending packet P1. Packet P2 is sent before the output O1 arrives from the

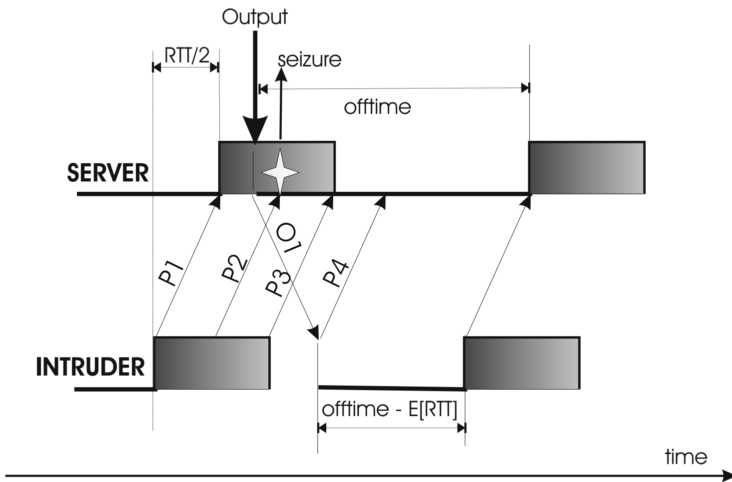

Fig. 6. Permanent phase

server. After that, the packet P2 seizes the new free position appeared after the response (O1). However, the intruder continues the *ontime* period till the reception of the response (packet O1). Meanwhile, the intruder sends the packet P3, which will cause an overflow event in the server upon its reception. On the sending of this last packet during the *ontime* period, the intruder will schedule a timer with value *offtime* and the new period begins. But, on the reception of the answer to packet P1 (packet O1), the period stops and a new one is started.

This mechanism is suitable to achieve our goal. Nevertheless, there is another case to consider. What happens if the output is raised after the reception of the whole ontime period in the server? In this case, the free position would be available till the next *ontime* to be seized by a user. This implies a high probability of failure in the aim of seizing the free position. To avoid this effect, on the reception of an answer from the server, the intruder will send another attack packet (packet P4 in Fig. 6) and reschedule the period with $offtime - E[RTT]$. Obviously, when the intruder does not receive the response (e.g., the output in the considered period corresponds to a request made by a legal user instead of the intruder) and the output is raised after *ontime*, it is more probable to fail the seizure. Thus, we can conclude that seizure failures, after a delay due to the queue time, increase the probability of new failures.

3.2 Transitory Phase Execution

The aim of the transitory phase is to initially flood the service queue. This phase could be made by high rate flooding, although this would make the attack more detectable. However, if the rate of the flooding is slightly higher than the service rate in the server, the queue will end up flooded after some periods. That is, initially the attacker will acquire some positions in the service queue. If the methods associated to the permanent phase are applied, those positions will remain captured, while the excess of traffic would eventually add seized positions from the intruder. This way, the attacker will finally get the whole queue full of his/her requests.

Usually, the intruder will detect the complete flooding of the service queue by means of the observation of the rejection of his/her packets. For example, in a connection oriented application the reception of RESET messages answering the SYN requests (in the case of TCP connections) will identify the complete flooding.

4 Attack Evaluation

In order to check the efficiency of the attack, and to validate the current work, it is convenient to establish some indicators to measure its performance:

- The *percentage of seizures* (S) as the number, in percentage, of buffer positions seized by the attack divided by the total number of seizures experienced in the server, within a specific period of time. This is a measure of the efficacy of the attack.

- The *effort of the attack* (E) as the number, in percentage, of MO packets received by the intruder, divided by the total number of packets generated by the intruder. This value represents the effort made by the attacker.
- The *user success percentage* (U) as the ratio, in percentage, between the number of seizures made by a legal user and the total number of packets generated by him/her. U is a measure of the perception on the service level experienced by a given legal user.
- The *overflow percentage* (O) as the ratio, in percentage, of the total MO messages divided by the number of packets generated by the server. This parameter tells about the saturation experienced by the server.

Among all the indicators, only the effort of the attack is observable by the intruder during the attack. This is because only the server knows the total number of packets and seizures generated during the observation period, which are necessary figures to calculate S and O. On the other side, U will be observable only by the legal users that try to access to the server by usual methods.

The aim of the attack is to minimize U, that is, the user perception of the availability of the server. This is similar to maximize S, the more the server is engaged more time with intruder requests, the more the user success percentage decreases. Besides, in order not to be detected by any active intrusion detection system, the attack should also minimize its effort, E. On the other hand, minimizing E will contribute to a lower overflow O in the server, making thus the attack more undetectable.

In order to evaluate the performance and effectiveness of the attack, we have implemented within Network Simulator (NS2) [22], a new object derived from the application class that behaves like the intruder (*intruder class*) and implements the two phases specified in Section 3.

We are interested in discovering how much effective the low rate DoS attack could be. A set of attacks has been tested in the simulator, and worrying results have been obtained because of the effectiveness constated. Fig. 7 shows the results obtained from an example of one attack simulation composed by 1332 outputs. The attack has been launched against a server with $T_s = \mathcal{N}(3.0\,s, 0.2)$ and $E[T_a] = 4.0\ s$. The parameters of the attack have been tuned choosing values for *ontime* = 0.6 s, *offtime* = 2.7 s, *offset* = 0 s, and *interval* = 0.3 s.

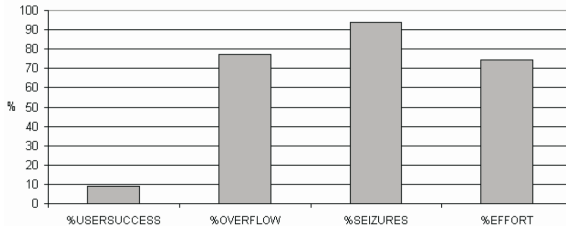

Fig. 7. Effectiveness for one attack

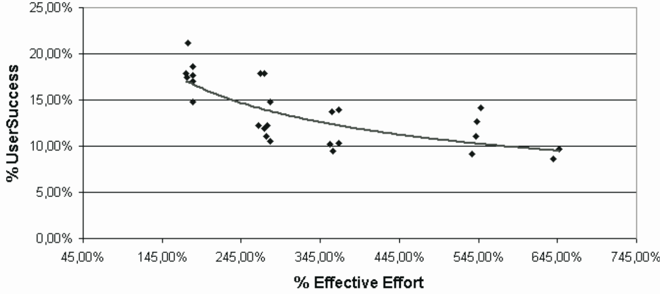

Fig. 8. User success and effective effort for 25 different configurations of the low-rate DoS attack

Additionally, the round trip time has been set to $RTT = \mathcal{N}(0.6\,s, 0.2)$ and the capacity of the system is $N = 20$.

It can be observed that a very high efficiency is obtained, given by $S = 94\%$ and $U = 9.04\%$. On the other side, the value of the *overflow percentage*, $O = 77.08\%$, indicates that the traffic offered to the server by both the users and the intruder is four times its capacity. If the intruder wanted to bypass an IDS system able to detect attacks on this rate, effectiveness should be sacrificed to reduce the overflow rate. This could be easily achieved by reducing *ontime* or increasing *interval*. In order to know the amount of traffic that the intruder generates, the *effective effort* is defined as the traffic offered by the intruder normalized by the capacity of the server:

$$\text{effective effort} = \frac{\text{packets sent by intruder}}{\text{total packets accepted by server}} \tag{6}$$

In Fig. 8, 25 possible attacks to the previously defined server are shown. The user success percentage and the effective effort are represented in main and secondary axes, respectively. It can be seen that there are a lot of possible configurations eligible for the attack in order to bypass IDS systems and, at the same time, obtain the best efficiency. As it has already been deduced in previous sections, a better efficiency implies a higher effort, as it can be seen in the tendency line.

Although these values could indicate a minor effectiveness, the results give us a good idea about the performance of the attack.

5 Real Environment Validation

We have tested also the proposed attack in a controlled real application to check its validity. The selected server is a web server that keeps the condition of serving requests in a monothreaded way. Although this is not a typical monothreaded application, it can be configured this way for this purpose. Former extension of the attack to multithreaded servers has motivated the election

Table 1. Real time and simulation values for selected experiments

E[Ts]	E[Ta]		*U*	*O*	*S*	*E*
3	3,5	**Simulated**	10,4	71,4	90,5	65,3
		Real	9,8	69,4	91,4	61,9
5	6	**Simulated**	5,7	67,7	94,2	55,8
		Real	7,8	67,6	92,5	56,5
10	12	**Simulated**	3,0	64,4	97,2	51,0
		Real	6,4	65,5	94,3	53,2
15	17	**Simulated**	3,0	66,6	96,8	51,0
		Real	2,5	65,5	97,7	50,7
20	22	**Simulated**	3,0	65,0	97,2	50,8
		Real	4,3	65,1	96,0	51,1
25	28	**Simulated**	1,8	64,6	98,0	50,5
		Real	1,8	65,4	98,3	50,3

of this real environment. Therefore, Apache 2.0.52, configured with the directive `ThreadsPerChild = 1`, presents a monoprocess behaviour. This way, we ensure that the processing of a request blocks those awaiting in the service queue. Also, we have considered that every request consists in a connection request. The attack establishes connections and sends no messages on them, letting the web server to close the connection after a timeout period specified by the Apache directive `Timeout`. This directive corresponds in our model to the value $E[T_s]$ in the server configuration value.

The scenario is analogous to that one used for the theoretical study. The user traffic has been generated following a Poisson process. A piece of software launches the attack from a single source. Both legal users and intruder traffic flows traverse a WAN network to reach the server, with a round trip time $RTT = \mathcal{N}(17ms, 0.05ms)$. Traces on the users and the intruder side have been issued for collecting the necessary data to calculate the attack indicators.

Among the results obtained from the set of experiments carried out, Table 1 shows some examples with a comparison between the obtained indicators in the simulation and in the real environment, for different mean service times ($E[T_s]$) and user traffic arrival rates ($E[T_a]$). The values for $E[T_a]$ have been selected in such a way that there is no congestion on the server without any attack. In these experiments, we have chosen values for *ontime*= 0.4 s, *interval* = 0.4 s, *offset*= 0, and *Var[*T_s*]*=0.01. The capacity of the server is $N = 7$.

We can observe the accuracy of the results obtained from simulations when compared with those for the real environment. Surprisingly, we can even obtain better results in efficiency (lower U and higher S, with lower O) for the attack in a real environment in some cases (e.g., first scenario with $E[T_s] = 3$). From these results, two conclusions can be made: a) all the experiments in the simulation case seem to provide results that are good approximations to the behaviour in real environments, and b) the real impact of the attack is very high, showing that these vulnerabilities in monoprocess servers are able to be easily exploited.

6 Conclusions

In this work, a vulnerability present in monoprocess or monothreaded servers is described. It consists in the possibility of knowledge, by an intruder, of the inter-output time of a server under congestion conditions. This vulnerability allows an intruder to perform a denial of service attack against a server. The attack could be designed to nearly or completely saturate the capacity of the target system but, as a difference from the usual brute-force DoS attacks, it uses a relatively low rate of packets to the target server to achieve its goal. Moreover, it is possible to tune the attack parameters in order to select the appropriate values for efficiency and load generated in the server. This distinctive characteristic could allow the attack to bypass, in many cases, existent IDS systems, thus becoming a non-detectable threat.

As a difference to other existent low rate DoS attack [15], this one threatens the application level, maintains the server engaged serving intruder requests and get advantage of the knowledge of the inter-output time of the target server. However, it has some common features with the low rate DoS attack to TCP, what points out the existence of a new family of DoS attacks, characterized by the fact that they rely on vulnerabilities that consist in the a-priori knowledge of one timer of a protocol or end-system behaviour, and that allows the intruder to carry out the DoS attack with a low rate of transmission, avoiding most of the current IDS systems.

The fundamentals and details of the design of a possible exploit have been explained. It has been demonstrated that this attack can be easily carried out and that it can obtain very efficient results. The potential risk that this attack presents is really worrying, due to the fact that it could behave very similar to legacy users, bypassing IDS systems and possibly affecting the services in a server.

As a future work, we plan to extend the study of this vulnerability in concurrent servers, due to the fact that this kind of servers is more widely deployed in Internet.

References

1. M. Williams. Ebay, amazon, buy.com hit by attacks, 02/09/00. IDG News Service, 02/09/00, http://www.nwfusion.com/news/2000/0209attack.html.
2. Jelena Mirkovic , Peter Reiher, A taxonomy of DDoS attack and DDoS defense mechanisms, ACM SIGCOMM Computer Communication Review, v.34 n.2, April 2004
3. CERT Coordination Center, Denial of Service attacks. Available from <http://www.cert.org/tech_tips/denial_of_service>
4. Computer Security Institute and Federal Bureau of Investigation, CSI/FBI Computer crime and security survey 2001, CSI, March 2001. Available from <http://www.gocsi.com>.
5. D. Moore, G. Voelker, S. Savage, Inferring Internet Denial of Service activity, in Proceedings of the USENIX Security Symposium, Washington, DC, USA, 2001, pp. 9-22.

6. P. Ferguson, D. Senie, Network ingress filtering: defeating Denial of Service attacks which employ IP source address spoofing, in RFC 2827, 2001.
7. Global Incident analysis Center - Special Notice - Egress filtering. Available from <http://www.sans.org/y2k/egress.htm>.
8. X.Geng, A.B.Whinston, Defeating Distributed Denial of Service attacks, IEEE IT Professional 2(4)(2000) 36-42.
9. N.Weiler, Honeypots for Distributed Denial of Service, in Proceedings of the Eleventh IEEE International Workshops Enabling Technologies: Infrastructure for Collaborative Enterprises 2002, Pitsburgh, PA, USA, June 2002, pp. 109-114.
10. Axelsson S. Intrusion detection systems: a survey and taxonomy. Department of Computer Engineering, Chalmers University, Goteborg, Sweden. Technical Report 99-15; March 2000.
11. R.R.Talpade, G.Kim, S.Khurana, NOMAD: Traffic-based network monitoring framework for anomaly detection, in Proceedings of the Fourth IEEE Symposium on Computers and Communications, 1998.
12. J.B.D. Cabrera, L.Lewis, X.Qin, W.Lee, R.K.Prasanth, B.Ravichandran, R.K.Mehra, Proactive detection of Distributed Denial of Service Attacks using MIB traffic variables - a feasibility study, in Proceedings of the 7th IFIP/IEEE Internation Symposium on Integrated Network Management, Seattle, WA, May 14-18, 2001.
13. J. Mirkovic, G.Prier, P.Reiher, Attacking DDoS at the source, in Proceedings of ICNP 2002, Paris, France, 2002, pp. 312-321.
14. DDoS attacks and defense mechanisms: classification and state-of-the-art, in Computer Networks 44, 2004, pp. 643-646.
15. A. Kuzmanovic and E. Knightly, Low Rate TCP-targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants), in Proc. ACM SIGCOMM 2003, Aug. 2003, pp. 75-86.
16. H.Sun, J.C.S. Lui, and D.K.Y.Yau, Defending Against Low-Rate TCP Attacks: Dynamic Detection and Protection, in Proc. IEEE Conference on Network Protocols (ICNP2004), Oct. 2004, pp. 196-205.
17. G. Yang, M. Gerla, and M. Y. Sanadidi, Randomization: Defense Against Low-rate TCP-targeted Denial-of-Service Attacks, in Proc. IEEE Symposium on Computers and Communications, July 2004, pp. 345-350.
18. A. Shevtekar, K. Anantharam and N. Ansari, Low Rate TCP Denial-of-Service Attack Detection at Edge Routers, in IEEE Communications Letters, vol 9, no. 4, pp. 363-365, April 2005.
19. SANS Institute. NAPTHA: A new type of Denial of Service Attack, December 2000. http://rr.sans.org/threats/naptha2.php
20. Roberta R. Martin. Basic Traffic Analysis. Prentice-Hall Inc. September 1993. ISBN: 0133354075.
21. http://mathworld.wolfram.com/ExponentialDistribution.html
22. Network Simulator 2.Available at: http://www.isi.edu/nsnam/ns/