International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 12, December - 2013

On A Recursive Algorithm for SYN Flood Attacks

Pranay Meshram[1], Ravindra Jogekar[2], Pratibha Bhaisare[3]
[123]*Department of Computer Science and Engineering*
[12]*Priyadarshini J L College of Engineering,* [3]*Abha-Gaikwad Patil College of Engineering*
[123]*RTM Nagpur University, Nagpur*

Abstract

A Denial of Service (DoS) attack is a generic term for a type of attack, which can take many forms. Machines that provide TCP services are often susceptible to various types of Denial of Service attacks from external hosts on the network. It can be characterized as an explicit attempt by attackers to prevent legitimate users of a service from using that service. Our main aim is to implement a defence mechanism for SYN flood attack on a network using OMNeT++. Finally, we compare OMNeT++ with NS-2 and propose OMNeT++ as better simulation software

Keywords: *DoS, SYN Flood attacks, OMNeT++.*

1. Introduction

A normal outcome of the vulnerability of TCP protocol, i.e. handshaking mechanism, SYN flood attacks are one of the most common DoS attacks which may halt the services provided by a server. It is always better to state the views on a rough paper than to implement it beforehand. OMNeT++ provides us with the all the tools required for modelling of wired and wireless communication networks, protocols, queuing networks, multiprocessors and other distributed hardware systems, validating of hardware architectures, and in general, it can be used for the modelling and simulation of any system where the discrete event approach is suitable, and which can be conveniently mapped into entities communicating by exchanging messages.

1.1 Denial of Service attacks:

A Denial of Service (DoS) attack is a generic term for a type of attack, which can take many forms. The motivation for DoS attacks is not to break into a system but to make the target system deny the legitimate user giving service. There are three basic types of attack, destruction or alteration of configuration information, physical destruction or alteration of network components, and consumption of scarce, limited, or non-renewable resources [1].

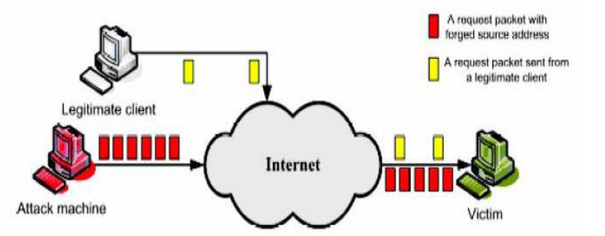
A request packet with forged source address
A request packet sent from a legitimate client
Legitimate client
Attack machine
Internet
Victim

Figure 1: Denial of service attack

i. Resource consumption

A well-known attack method is the "SYN flood" [4], which exploit vulnerability in the TCP protocol. A target machine is flooded with TCP SYN packets. The source TCP ports and source addresses of the request packets are randomized in order to make it look like each packet correspond to a new request for a TCP connection, but the connection will never be completed. The purpose is to force the target host to maintain state information for every request and eventually run out of available TCP connections. This type of attack has been known for some time and many operating systems have increased their buffer space for "half-open" TCP connections to become more resilient to this type of attack. Another attack type called "stream" [5] uses TCP ACK packets. As in SYN flood, port and source address is randomized. It has shown to be quite effective, especially if it hits an open port. The effectiveness is highly dependent on how the operating system handles TCP ACK packets with no corresponding connection. Email bombing and spamming [6] can also be used to consume system resources.

ii. Bandwidth consumption

A common goal of all bandwidth attacks is to consume all available bandwidth in a network. This can be achieved by sending more packets to the network than it can handle, called packet flooding. Preferably single, not connection oriented, spoofed IP packets are used. This makes UDP and ICMP packets best suited for flooding. Another possibility is to connect the chargen function on one machine to the echo function on another machine on the same network [7]. Ping can be used for flooding by starting up multiple copies of the program on a well connected server and using a command option to pad each ICMP echo request packet with lots of extra data [8].

iii. Distributed DoS attacks

A more recent and well-known attack called "smurf" [9] use reflectors to multiply the effect of the attack. Instead of sending the flood of ICMP echo requests directly to the victim it is sent to the broadcast address of another network. The address of the victim machine is included as the source address in the packet and causes every machine on the network to respond to this machine pts.

2. Normal three way handshake in TC

As we know, a connection needs to be established between the source S and destination D to facilitate the communication between them. This process is referred as the three-way handshake. The process starts with the source sending a SYN packet (TCP header with SYN bit set) to D who responds by sending back packet with both SYN and ACK bits set. If the source finally responds with ACK bit set, connection is established else D sends RST signal after timeout period. Three-way handshake is also used for initializing the sequence numbers, which are needed to provide reliable delivery of packets.

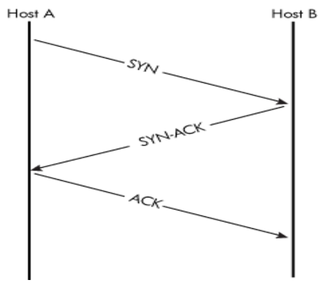

Figure 2: Three-way handshake

Three memory structures namely socket structure (socket), internet protocol control block structure (inpcb) and the TCP control block structure (tcpcb) are allocated by both S and D for every connection. These structures contain all the information required for the connection like state information, buffers, address information, flags, timer information, port numbers, sequence number information etc.[11] Thus, for every connection, certain memory resource is consumed and ports are utilized. This is the main purpose of the attacker.

3. SYN Flood attack:

The SYN flood attack is, simply, to send a large number of SYN packets and never acknowledge any of the replies. This leads the recipient to accumulate more records of SYN packets than his software can handle. Flooding attacks intend to overflow and consume resources available to the victim (memory, Bandwidth) by sending a continuous flood of traffic. SYN flooding is the most common and well known DoS attack. In SYN flooding, the attacking system sends SYN request with spoofed source IP address to the victim host. These SYN requests appear to be legitimate. The spoofed address refers to a client system that does not exist. Hence final ACK message will never sent to the victim server system. This results into more number of half-open connections at the victim side. A backlog queue is used to store these half-open connections. These half-open connections bind the resources of the server. Hence no new connections (legitimate) can be made, resulting in Denial of Service [4].

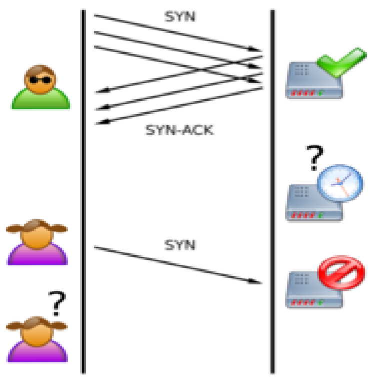

Figure 3: SYN Flood attack

Defence mechanisms for SYN Flood attacks:

Using firewalls:

Firewalls can protect against SYN flooding attacks using two methods:

Firewall as a Relay:

Firewall acts on behalf of the host like a proxy. When SYN message arrives, the firewall sends the message SYN, ACK to the source. D is not aware of any packets at this time. If the final ACK does not arrive, as would be in case of an attack, the firewall resets the connection and the host doesn't receive any packets.

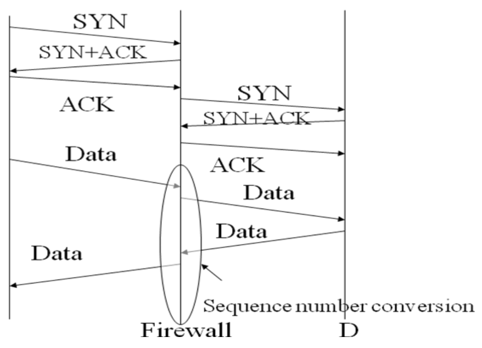

Figure 4: Firewall as a Relay(legitimate connection request)

On the other hand, if the connection is legitimate and the firewall receives the third message of ACK, it forms a new connection with the host on behalf of the actual client. From here on, the firewall has to change the sequence numbers in the packets from client to server and send them to server [12]. This results in an additional delay of messages for legitimate connections. This is an effective technique only if the firewall is resistant to SYN flooding attack.

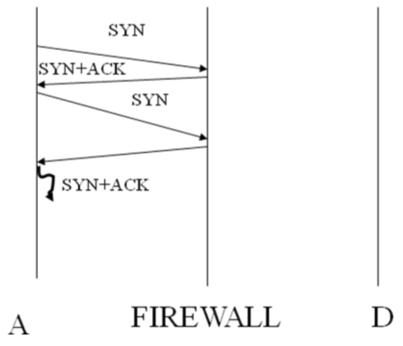

Figure 5: Firewall as a Relay (Illegitimate connection request)

Firewall as a Semi-transparent Gateway:

The firewall monitors the traffic sent from source to destination. When it sees ACK+SYN being sent from D to S, it responds by creating an ACK message and sending to D, thus reallocating the resources and moving the connection out of the queue. If it is an attack, the firewall then sends a RST message to D and connection is dropped.

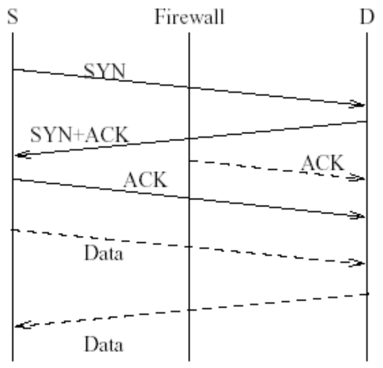

Figure 6: Firewall as a Semi-transparent Gateway
(Legitimate connection request)

If the connection request is from a proper source, he sends back ACK, which is passed by firewall. D merely sees it as the duplicated packet and discards it. This prevents the delay that was introduced in earlier method. The only drawback is that there are many open connections at D in case of an attack. [13]

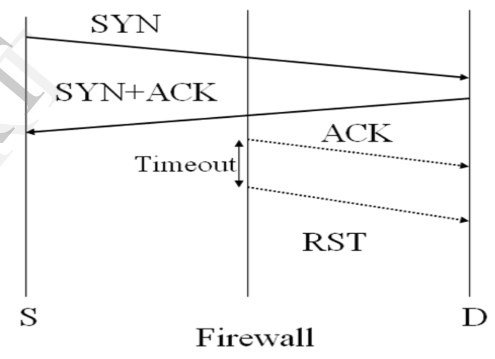

Figure 7: Firewall as a Semi-transparent Gateway
(Illegitimate connection request)

4.An overview of OMNeT++

OMNeT++ is a C++-based discrete event simulator for modelling communication networks, multiprocessors and other distributed or parallel systems.

4.1 Model structure:

An OMNeT++ model consists of modules that communicate with message passing. The active modules are termed simple modules; they are written in C++, using the simulation class library. Simple modules can be grouped into compound modules and so forth; the number of hierarchy levels is not limited. When a module type is used as a building block, there is no distinction whether it is a simple or a compound module. This allows the user to transparently split a module into several simple modules within a compound module, or do the opposite, re-implement the functionality of a compound module in one simple module, without affecting existing users of the module type.

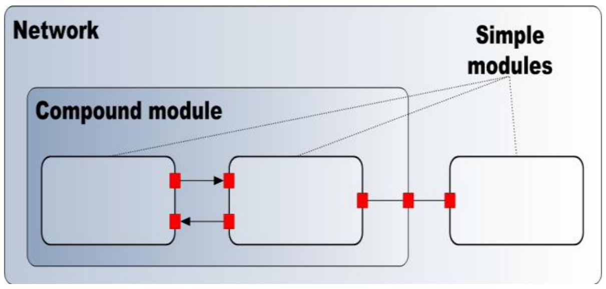

Figure 8: Modules in OMNeT++

Modules communicate with messages which – in addition to usual attributes such as timestamp – may contain arbitrary data. Simple modules typically send messages via gates, but it is also possible to send them directly to their destination modules. Gates are the input and output interfaces of modules: messages are sent out through output gates and arrive through input gates. An input and an output gate can be linked with a connection. Connections are created within a single level of module hierarchy: within a compound module, corresponding gates of two submodules, or a gate of one submodule and a gate of the compound module can be connected. Connections spanning across hierarchy levels are not permitted, as it would hinder model reuse. Properties such as propagation delay, data rate and bit error rate, can be assigned to connections. One can also define connection types with specific properties (termed channels) and reuse them in several places.

Modules can have parameters. Parameters are mainly used to pass configuration data to simple modules, and to help define model topology. Parameters may take string, numeric or boolean values.

4.2 The Design of the NED Language

The user defines the structure of the model (the modules and their interconnection) in OMNeT++'s topology description language, NED. Typical ingredients of a NED description are simple module declarations, compound module definitions and network definitions. Simple module declarations describe the interface of the module: gates and parameters. Compound module definitions consist of the declaration of the module's external interface (gates and parameters), and the definition of submodules and their interconnection. Network definitions are compound modules that qualify as self-contained simulation models. In addition to a number of smaller improvements, the following major features have been introduced:

Inheritance: Modules and channels can now be subclassed.Derived modules and channels may add new parameters, gates, and (in the case of compound modules) new submodules and connections.

Interfaces: Module and channel interfaces can be used as a Placeholder where normally a module or channel type would be used and the concrete module or channel type is determined at network setup time by a parameter.

Packages: To address name clashes between different models and to simplify specifying which NED files are needed by a specific simulation model, a Java-like package structure was introduced into the NED language.

Inner types: Channel types and module types used locally by a compound module can now be defined within the compound module, in order to reduce namespace pollution.

Metadata annotations: It is possible to annotate module or channel types, parameters, gates and submodules by adding properties. Metadata are not used by the simulation kernel directly, but they can carry extra information for various tools, the runtime environment, or even for other modules in the model. For example, a module's graphical representation (icon, etc) or the prompt string and unit (mill watt, etc) of a parameter are specified using properties.

5. Methodology for SYN flood attack on OMNeT++

Our first module of the project for simulating a SYN flood attack on OMNeT++ includes the following stages:

- Initiating a SYN flood attack
- Detecting the attack

Our second module of the project deals with the defense mechanism implemented for prevention. Here, we discuss the first module in order to describe structure of NED language.

First module:

For any project to begin, we have to create certain modules which will act as classes for OMNeT++. These modules are actually the description about our nodes which we are going to use in our Network simulation. Our project consists of 3 types of modules. These are attacker module, the server and a legitimate host, to name. These modules have certain parameters and gates associated with them to send and receive messages. The whole Network description is given in the NED file.

Now, SYN flooding is a type of DoS attack, wherein the attacker sends continuous SYN messages to the server and gets the acknowledgement every time as SYN+ACK. But this attacker is not bothered about sending the ACK packet to the server back. It just continuously goes on sending SYN packets. After a certain number of SYN packets being sent to the server (we have set the queue limit to 6), the server queue gets full and flooding occurs with the server left with half-open connections. The attacker still attacks the server with SYN packets continuously.

Now, the legitimate host tries to establish a connection with the server, by sending a SYN packet and finds that the server cannot be accessed. This is because the server queue has got full to accommodate any further connection request. So, it rejects any further request. Before the flooding took place, if the remote host had sent this same connection request, the server would have acknowledged it. But, after flooding, it is not possible till the queue again gets emptied.

After a certain timeout period, the queue will get refreshed and it will welcome any new connection request again till the queue again gets full. Whenever the connection is established, that particular entry is removed from the queue.

For every module of the OMNeT++ simulation, following two functions have to be defined:

- initialize() includes building the model and inserting initial events to FES(Future Event Set).
- handleMessage() is a method that is called by the simulation kernel when the module receives a message.

The following algorithm along with some functions presents an overview of the structure of OMNeT++ programming language called NED. The following algorithm is developed for SYN flood attack initiation and detection:

Algorithm For SYN flood attack initiation and detection:

```
Void attacker:initialize()
{       Define event type sendself1 of type cMessage;
        Send self msg at simtime()=0.0;
}

void attacker:handlemessage(cMessage *msg1)
{
        if(msg1 kind=sendself1)
        {
                Create new  cMessage m1 of type
SYN1;
                Send(m1,server);

        scheduleAt(simTime()+delay,sendself1);
        }
        else
                Delete msg;
}
Void server:initialize()
{
        Define event type timeout;
        scheduleAt(simTime()|+refresh, timeout);
}
Void server:handleMessage(cMessage *msg)
{
        If(count>queue length)
        {
                Display"SYN Flood attack";
                Set sign;
        }
        If(msg kind=SYN1 && (!sign))
        {
                Create new cMessage m1 of type
SYN+ACK;
                Send(m1, attacker);
                Delete msg;
        }
        If(msg kind=timeout)
        {
                Refresh queue;
        }
        If(msg kind=ACK2)
        {
                Free queue by 1 slot;
                Delete msg;
        }

}
Void legitimate:initialize()
{
        Define event type sendself2 of type cMessage;

        Send self msg at some time=t;
}
void legitimate:handlemessage(cMessage *msg2)
{
        if(msg2 kind=sendself2)
        {
                Create new  cMessage m of type
SYN2;
                Send(m,server);
        }

        If(msg2 kind=SYN+ACK)
        {
                Create new  cMessage m2 of type
ACK2;
                Send(m2,server);
        }
        else
                Delete msg;
}
```

(Note: Sign and count are initially set to 0)

Here, cMessage is a message class in OMNeT++.
Some other functions used are:

- simTime(): simulation time
- scheduleAt(): used to schedule the events
- send(): to send messages to other modules

5. Conclusion

In this paper an algorithm has been implemented to simulate a SYN flood attack. The programming approach is quite simple to implement. Reducing the time out period or increasing the number of half open connections could not provide with the solution for preventing a system from getting SYN Flooded. Thus, the new concept of firewall came into existence. The OMNeT++ approach significantly differs from that of NS-2, J-Sim or any other simulation tools. While the NS-2 (and NS-3) project goal is to build a network simulator, OMNeT++ aims at providing a rich simulation platform, and leaves creating simulation models to independent research groups.

6. References

[1] Helena Sandström "A Survey of the Denial of Service Problem"
[2] Computer Emergency Response Team. “Results of the Distributed-Systems Intruder Workshop” http://www.cert.org/reports/dsit_workshop-final.html
[3] CNN.com, “The denial-of-service aftermath”
[4] Computer Emergency Response Team. CERT Advisory CA-96.21;“TCP SYN Flooding and IP Spoofing Attacks”;1996.
[5] BUGTRAQ threads on the stream.c DoS attack and its Fallout http://staff.washington.edu/dittrich/misc/ddos/stream.txt
[6] Computer Emergency Response Team, Tech Tips, “Email Bombing and Spamming” “http://www.cert.org/tech_tips/email_bombing_spamming.html”
[7] Computer Emergency Response Team. CERT Advisory CA-1996-01, “UDP Port Denial-of-Service Attack”,
[8] Computer Emergency Response Team. CERT Advisory CA-1996-26, “Denial-of-Service Attack via ping”,http://www.cert.org/advisories/CA-1996-26.html
[9] Computer Emergency Response Team. CERT Advisory CA-1998-01, “Smurf IP Denial-of-Service Attacks“,
[10]Jonathan Lemon "Resisting SYN flood DoS attacks with a SYN cache", Free BSD project
[11]http://www.phrack.org/issues.html?issue=48&id=13
[12]Proceedings of the 10th USENIX Security Symposium, Washington, D.C., USA, 2001
[13]The Internet Protocol Journal - Volume 9, Number 4-“Defenses Against TCP SYN Flooding Attacks”
[14]Andras Varga "An overview of the OMNeT++ simulation environment", 2008, France