DE GRUYTER OPEN

Print ISSN: 1312-2622; Online ISSN: 2367-5357
DOI: 10.1515/itc-2017-0020

# Network Simulator for Botnet DoS Attacks

**Y. Aleksieva, H. Valchanov**

***Key Words***: *Botnet; DoS attacks; network security; network attacks simulator.*

***Abstract***. *The most serious appearance of modern malware is Botnet. Botnet is a fast growing problem that is still not well understood and studied. One widely spreading variant of the botnet attacks is an IRC botnet. There are a number of techniques to detect botnet attacks and infections, but they do not have the functionality to prevent malicious botnet activity. The development of a quality equipment to detect and remove botnet infection needs a good simulation environment that is safe and fully meets the functionality of botnet DoS attacks. The paper presents the features of the implementation of network simulator for DoS attacks. The simulator allows creating different attacks with various parameters. Some performance experiences are performed and the obtained results are shown.*

## Introduction

The most serious manifestation of modern malware is Botnet. Botnet is a fast growing problem that is still not well understood and studied. In time botnets have evolved from simple generators of spam to primary means of mass criminal operations. Botnets are involved in identity and credit card numbers thefts, various forms of espionage, Denial of Service (DoS – denial of service) attacks, attacks on unsafe products from the new digital world – putting in danger the modern society and the evolving and expanding business [1].

Bot is a machine on which a malicious computer software has been installed and it can be controlled independently and automatically. Botnet is a collection of compromised computers running malware, being controlled by a hacker – commonly known as the botmaster, through a centralized infrastructure for command and control (C&C center) of the attack (*figure 1*). The C&C infrastructure controlling the bots is malicious in nature and may illegally control the computing resources of the bots. Botnet gains computers and turns them into an army of cyber attack to be used for spam, fake Web sites, DoS attacks, viruses, worms, backdoors, collecting information through phishing and fraud.

Botnet used with combination of modern technology such as Internet Relay Chat (IRC), Hypertext Transfer Protocol (HTTP) and Peer-to-Peer (P2P) allows attackers to organize hidden tactics in legal custom applications, making detection of botnet attacks much more complex.

One widely spreading variant of the botnet attacks is an IRC botnet which has been considered to be extinct – until 2015 when returning with large-scale DoS attacks. As of this year, attacks have increased by 132% more than last year. The problem with increasing DoS IRC botnet attacks concerns modern computer-based business as the primary means of internal communication in large (Google, Yahoo, LinkedIn, Facebook), as well as in small business corporations is based on the use of IRC clients and channels. Information exchanged on these channels is confidential, thus making these channels good target for attack.

There are a number of techniques developed to detect botnet attacks and infections, but they do not have the functionality to prevent malicious botnet activity. This makes them insufficiently effective tools for defense because they

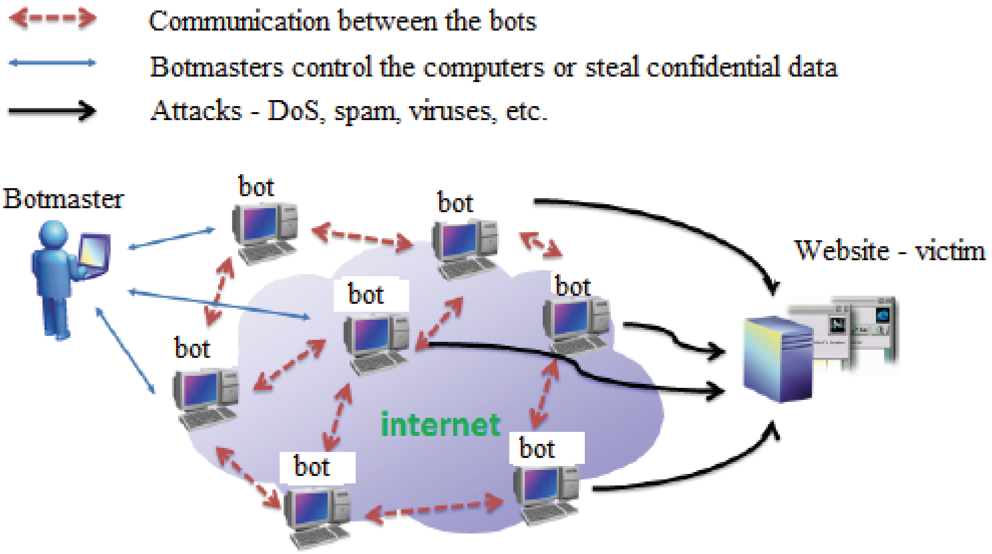

Communication between the bots
Botmasters control the computers or steal confidential data
Attacks - DoS, spam, viruses, etc.
Botmaster
bot
bot
bot
bot
bot
bot
bot
internet
Website - victim

**Figure 1.** Botnet action

are not comprehensive enough. The development of quality equipment to detect and remove botnet infection needs a good simulation environment that is safe and fully meets the functionality of botnet DoS attacks. The presence of such a simulation environment facilitates the development of protection techniques, as they can be tested, giving real results and will also give the ability to develop quality and effective means to prevent botnet attacks.

## Existing Solutions

Currently there are a variety of tools for generating various DoS flood attacks, but they are specialized in the implementation of a particular type of attack without the possibility of implementation of the other types. Few are the systems which generate various DoS attacks, and even less are those who generate botnet DoS attacks.

LOIC is one of the most popular tools for DoS attacks, freely available on the Internet [2]. It can be used only by a single user to carry out the DoS attack with target – small servers. It can realize UDP flood attack, TCP flood attack or HTTP requests to the server-victim. The disadvantage of this tool is that it has no option to realize DoS attacks like botnet, but uses only a single IP address and does not hide the IP address so it can be easily traced.

Another similar tool is XOIC [3]. It can realize (D)DoS attacks to a given IP address, as the user selects the port and protocol on which the attack to be carried on. Main application of this tool is to test the resilience of websites. The disadvantage is similar – realized DoS attacks are from one machine, not a botnet. Other thing is that this tool is not a simulator but a real system for security attacks, which makes it dangerous even only for testing purposes.

Alternative tool is DDOSIM [3]. This is a generator that implements HTTP flood and TCP flood attacks simulating the presence of several bot hosts with random IP addresses. Written in C++, it runs under Linux and is open source. It allows generation of attacks by a machine simulating the act is carried out by various hosts, which is not a real botnet simulation (it can use only the resources of the machine that is running to generate attacks).

The platform for testing MAZEBOLT implements flood attacks, botnet attacks and attacks at the application level [4]. The tool is paid and thus with closed source, which limits opportunities for its use and consequently suggesting improvements.

BoNeSi is a DDoS botnet simulator that simulates traffic in a closed environment to study the effects of this type of attack [5]. It generates ICMP, UDP and TCP flood attacks from botnet of a given size and different IP addresses. The simulator is open source, but is entirely console-controlled. Similarly to DDOSIM, it generates attacks from a machine simulating the act is carried out by various hosts. Another disadvantage is that the tool works only under operating system Linux.

In this paper are presented some aspects of the implementation of the simulation environment for generating IRC DoS network attacks. The simulator provides flexible tools with which can be simulated different types of attacks with various parameters both in real-time conditions as in a laboratory. The experimental studies of its functionality and results are presented.

## Botnet Attacks Nature

IRC botnet is the most widely spread botnet attack because it is simple to implement and relatively difficult to be detected the presence of attack. Botmaster hides behind several IRC servers and gives commands to attack to the bots via IRC channels in which the owners of the bot machines do not know they are present [6,7].

IRC is used as infrastructure C&C protocol through which the botmaster sends commands to his bot army. IRC is a chat system that provides one-to-one and one-to-many instant messaging over the Internet. The attacker can use public IRC server, or build his own server for the purpose of C&C. Bots are configured to connect back to the C&C server expecting on any port configured to receive commands and to implement it. The action of IRC botnet is shown on *figure 2*.

IRC botnet is used to perform DoS attacks, as the botmaster can easily manage attacks hidden behind IRC

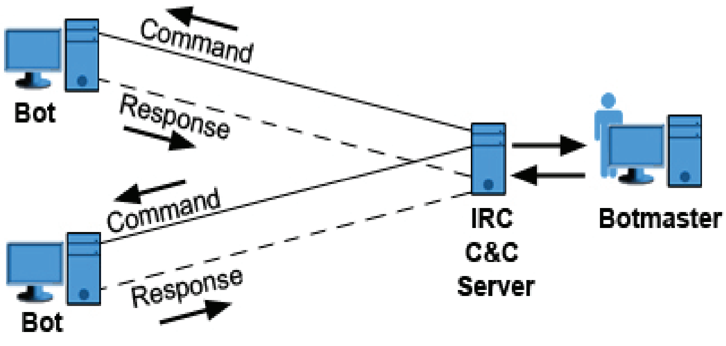


**Figure 2.** IRC botnet operation

technology and can hardly be detected. This attack is preferred because it is almost impossible to understand that a machine is part of a botnet and thus to be stopped because IRC botnet relies on a large number of zombie computers to realize small parts of the attack, but overall to get a large load. The capabilities of a single bot machine are not fully used when committing the attack and thus the resources of the bot machine are not much loaded. The aim of the IRC botnet is to have a large army of bots that simultaneously realize attack to a given victim. This way the owners of the bot machines can not know the presence of infection and to take actions to remove the bot from the network.

## Architecture of the Simulator

The architecture of the presented simulation environment has distributed nature. It consists of a number of bots running on separate machines and communicating through IRC channels. Each bot is built on a modular principle (*figure 3*). The use of modules allows for easy subsequent extension of the simulator with new components implementing additional classes of attacks.

Each bot has the following components:

- **Core** – implements the basic functionality of the bot, coordinating the work of its individual components.
- **Command interpreter** – realizes the inner system for accepting and processing commands from remote users.
- **Attack generator** – implements are the three basic types of attacks TCP, UDP and ICMP.
- Communicational subsystem – supports the communication to other bots in the network, using sockets.
- **IDE** – graphical interface of the bot, allowing interaction with the user of the local machine.

The presented simulation environment is designed to simulate the effects of malicious network attacks. Therefore, the method of converting machines in bots is not subject of research in the presented report. It is assumed that a botnet network is already established – computers were infected. In order to simulate the operation of IRC botnets, each machine must have installed a bot. Because of the possibility that software can be used for abuse, functionality for automatic start when starting up the machine is not added.

The system can operate with at least one bot connected to the network to an unlimited number of bots. Bots are started locally on each machine and are controlled by the IRC channels in which they join. This IRC channels represents the C&C center of the botnet. Management of the botnet DoS attacks is done remotely by sending commands in the specified IRC channels. All bots receive the same command and each of them executes it independently of the others in order to achieve more effective attack. After execution of the attack bots use the IRC protocol and connect to the user's channel that has sent the command. Then they send a message with a report on the work they have done. So the report is only visible to the attacker, but not for other users in the channel.

To implement the system in terms of research without violating the generally accepted laws, the IRC channel used is specifically designed for this purpose. This channel is not used by legitimate users of IRC technology. It is also used locally installed IRC server, not public, since the use of public IRC server would violate the etiquette of using the Internet and can lead to problems in actually used services. The locally installed server is unavailable for legitimate users of IRC communication technology, which keeps the security and the closed nature of the system.

Each bot can execute few types of commands received from the network:

- Commands for execution on the command interpreter shell.
- Commands for sending text message in the channel.
- Commands for starting TCP, UDP or ICMP attacks.
- Command for exiting the channel and suspending the bot action.

The need of combinability regarding the implementa-

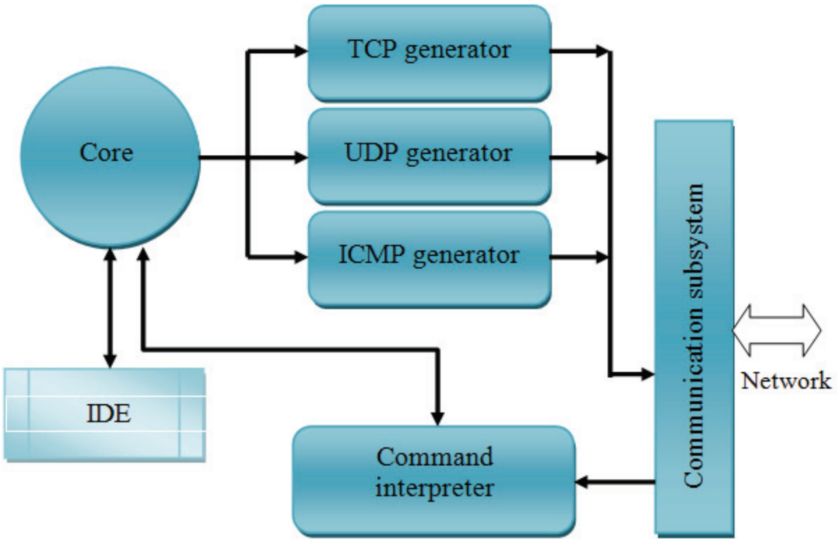


**Figure 3.** Architecture of the bot

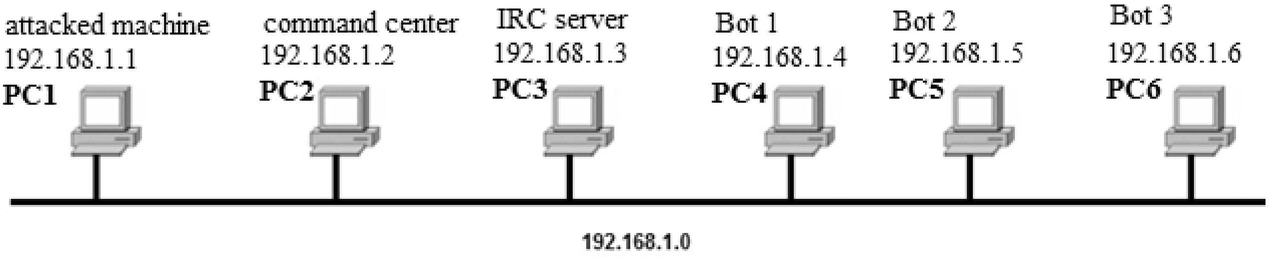


**Figure 4.** Flat topology

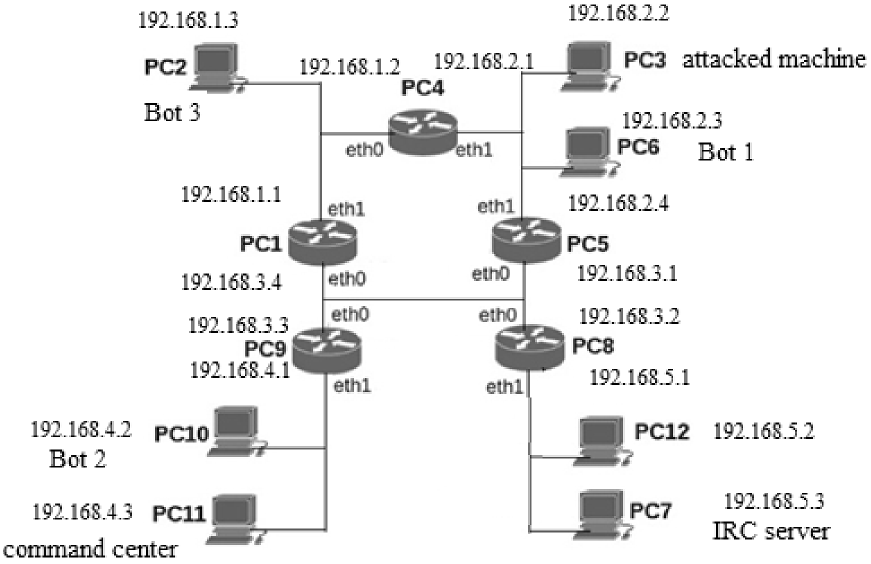


**Figure 5.** Routed topology

tion of various types of attacks requires the possibility that any type of attack to be performed autonomously with maximum options for parameterization. The parameters are names/hosts' addresses, port numbers, number of packages, size of the package contents and more. Each attack can be configured to endure certain length of time. After each command execution or attack bots send their status by personal message send to the botmaster. Each bot sends a message with a report with information as how many packages are sent, the duration of the attack and what is the result of the executed command.

## Experimental Study and Results

The testing of the simulation environment is made in terms of the two main types of network infrastructure – flat and routed.

In flat topology (*figure 4*) the bots, the command center, the IRC server and the victim machine are in one network. The objective is to determine the effects of the attacks and the quality of the bandwidth of the network with a single channel to the attacked machine.

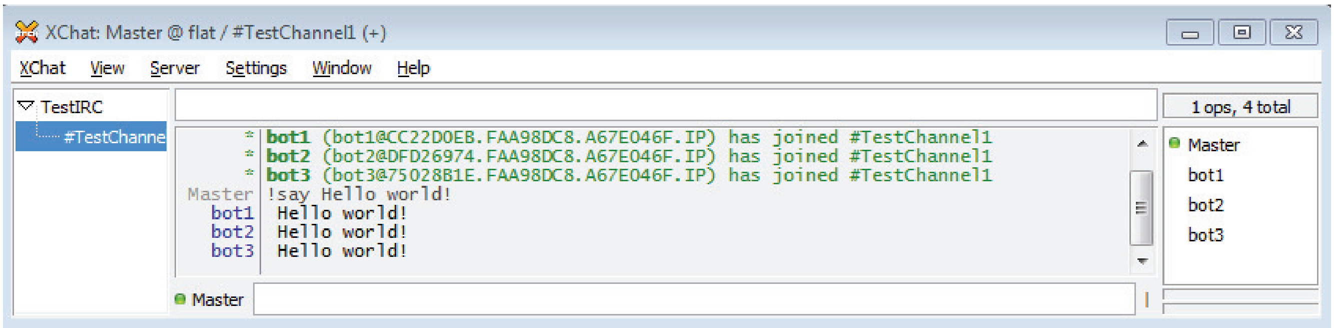
Two different experiments are made with the flat topology – one experiment with botnet containing 3 bots and other experiment containing 5 bots, as the other components of the network remain the same. The evaluation of the experiments is made according to two base comparisons – comparison between the results of the experiment with 3 bots in flat topology and 3 bots in routed topology, and the other comparison is done between the results of the experiments with 3 bots in flat topology and 5 bots also in flat topology.

In routed topology (*figure 5*), the individual components of the system are located in different subnets and routing protocols are used. The objective is to determine the effects of attacks and quality of the traffic capacity of the network in a real routed network environment.

The experimental platform is based on Intel Core2Duo T7700, running at 2.4 GHz clock speed and 4Gb RAM. The command center and the victim machine are running Windows 7 Enterprise, and the bots – Slackware Linux 3.10. For the purpose of the experiments was used UnrealIRCd IRC server which is open source and allows personalized configuration. This is by the fact that the use of a public IRC server would lead to conflict and violating the rules of the proper communication in IRC. For an IRC client is used the client Xchat.

The tests were performed in two directions: to test the communication between bots and the implementation of the network attacks. Bots run on the corresponding machines and at launch they join in the test channel through which the botmaster will send the commands.

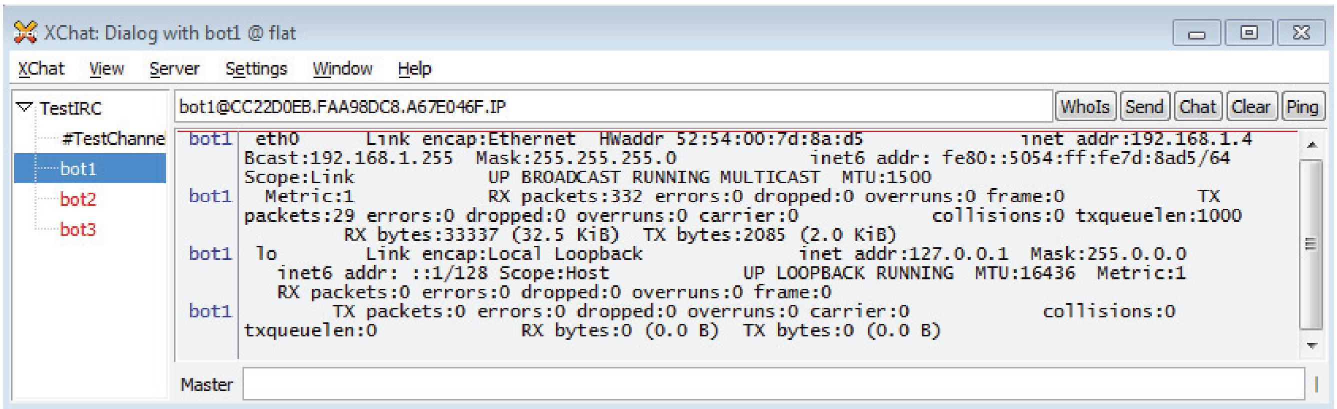
The first group of tests include testing of the connection of the bot to the botmaster by sending the command



XChat: Master @ flat / #TestChannel1 (+)
XChat View Server Settings Window Help
TestIRC
#TestChanne
1 ops, 4 total
Master
bot1
bot2
bot3
Master

```
      * bot1 (bot1@CC22D0EB.FAA98DC8.A67E046F.IP) has joined #TestChannel1
      * bot2 (bot2@DFD26974.FAA98DC8.A67E046F.IP) has joined #TestChannel1
      * bot3 (bot3@75028B1E.FAA98DC8.A67E046F.IP) has joined #TestChannel1
 Master !say Hello world!
   bot1  Hello world!
   bot2  Hello world!
   bot3  Hello world!
```

**Figure 6.** Joining in the channel and response



XChat: Dialog with bot1 @ flat
XChat View Server Settings Window Help
TestIRC
#TestChanne
bot1
bot2
bot3
bot1@CC22D0EB.FAA98DC8.A67E046F.IP
WhoIs
Send
Chat
Clear
Ping
Master

```
bot1  eth0      Link encap:Ethernet  HWaddr 52:54:00:7d:8a:d5           inet addr:192.168.1.4
     Bcast:192.168.1.255  Mask:255.255.255.0           inet6 addr: fe80::5054:ff:fe7d:8ad5/64
     Scope:Link             UP BROADCAST RUNNING MULTICAST  MTU:1500
bot1   Metric:1             RX packets:332 errors:0 dropped:0 overruns:0 frame:0            TX
     packets:29 errors:0 dropped:0 overruns:0 carrier:0           collisions:0 txqueuelen:1000
              RX bytes:33337 (32.5 KiB)  TX bytes:2085 (2.0 KiB)
bot1  lo        Link encap:Local Loopback             inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host            UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
bot1            TX packets:0 errors:0 dropped:0 overruns:0 carrier:0            collisions:0
     txqueuelen:0             RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

**Figure 7.** Execution of OS command

**!say**, followed by text. All the bots must repeat the text and to display it on the client's screen. *Figure 6* shows the results of the executed test with text "Hello world!".

The second test checks the capabilities of the bots to perform local operating system commands through the command **!cmd**. It requires as a parameter command of the OS. After the execution, the bots send the result in a personal message to the botmaster. Thus the result is visible only to the botmaster, but not for other users in the channel. The results of the test for the execution of the OS command *ifconfig* on a machine are shown on *figure 7.*

The tests show the functionality of the bots. It should be noted that the results are similar for the two types of network topologies for the three experiments that were made.

The second group of tests examines the functionality and efficiency of the generated network attacks. The objective is to determine the effectiveness of the simulation environment for congestion of the communication channel with packets. The attacks were performed on the three basic protocols.

The first type of attack is UDP flood determined to host on port 53 (DNS) with duration 120 s. The packages contain symbols '**X**'. *Figure 8* shows a report given by the network analyzer WireShark for the traffic to the attacked machine.

The tests show the functionality of the bots. It should be noted that the results are similar for the two types of network topologies for the three experiments that were made.

The second group of tests examines the functionality and efficiency of the generated network attacks. The objective is to determine the effectiveness of the simulation environment for congestion of the communication channel with packets. The attacks were performed on the three basic protocols.

The first type of attack is UDP flood determined to host on port 53 (DNS) with duration 120 s. The packages contain symbols '**X**'. *Figure 8* shows a report given by the network analyzer WireShark for the traffic to the attacked machine.

The ICMP attack includes the implementation of a number of ping commands from the command line of the machine by all the bots. The result of the attack is shown on *figure 10.*

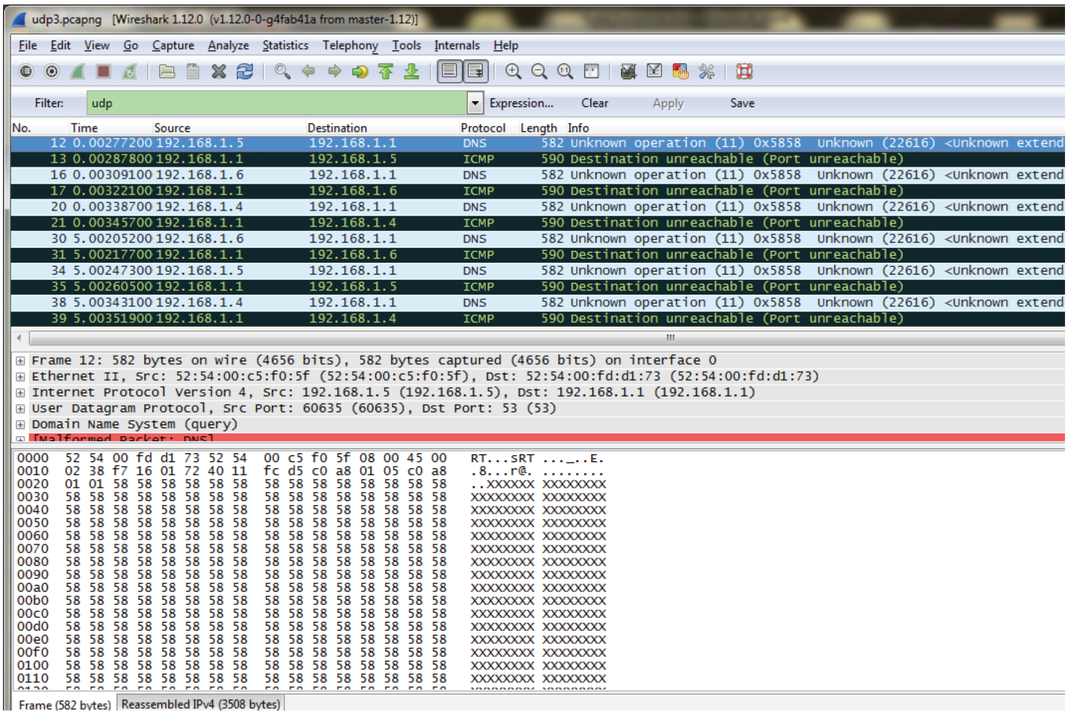
Information on the results of the traffic for each attack is derived from the summary report of WireShark, as shown on *figure 11.*

*Table 1* shows the average results of the experiments for both network topologies – flat and routed. The results are generated based on summary reports from the analyzer.

**Table 1.** Average traffic values for flat and routed topologies

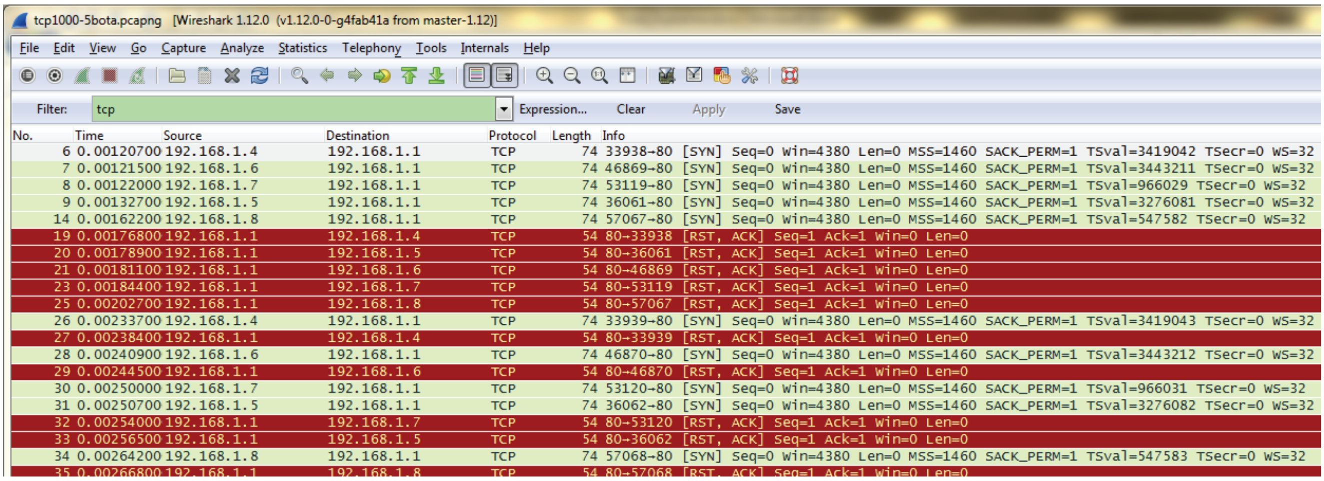
| Attacks | Packets/s | | Size (bytes) | | Bytes/s | |
|---|---|---|---|---|---|---|
| | Flat | Routed | Flat | Routed | Flat | Routed |
| UDP | 1.7 | 1.4 | 897 | 893 | 1550 | 1213 |
| TCP | 569 | 95 | 64 | 64 | 36407 | 6106 |
| ICMP | 3.4 | 3.3 | 98 | 96 | 335 | 321 |

For the average packet size there are no big differences in the values in both topologies. For the average number of packets per second and the number of bytes per second the values reported for UDP and ICMP attacks have a little gap, as in the second topology the values are


udp3.pcapng [Wireshark 1.12.0 (v1.12.0-0-g4fab41a from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: udp
Expression... Clear Apply Save
Frame (582 bytes)
Reassembled IPv4 (3508 bytes)

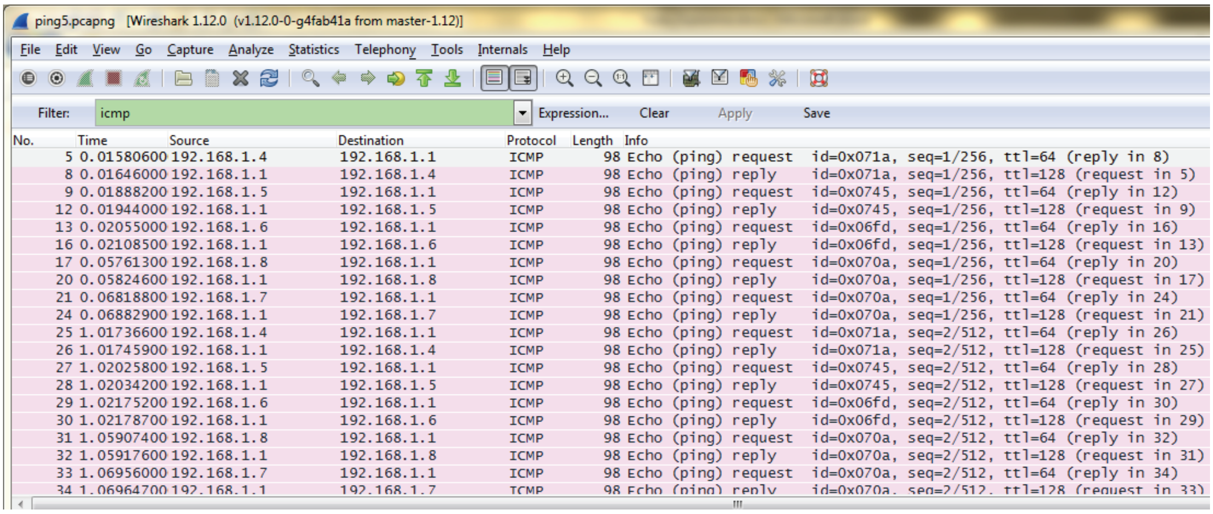
```
No.  Time       Source      Destination  Protocol Length Info
  12 0.00277200 192.168.1.5 192.168.1.1  DNS      582 Unknown operation (11) 0x5858  Unknown (22616) <Unknown extend
  13 0.00287800 192.168.1.1 192.168.1.5  ICMP     590 Destination unreachable (Port unreachable)
  16 0.00309100 192.168.1.6 192.168.1.1  DNS      582 Unknown operation (11) 0x5858  Unknown (22616) <Unknown extend
  17 0.00322100 192.168.1.1 192.168.1.6  ICMP     590 Destination unreachable (Port unreachable)
  20 0.00338700 192.168.1.4 192.168.1.1  DNS      582 Unknown operation (11) 0x5858  Unknown (22616) <Unknown extend
  21 0.00345700 192.168.1.1 192.168.1.4  ICMP     590 Destination unreachable (Port unreachable)
  30 5.00205200 192.168.1.6 192.168.1.1  DNS      582 Unknown operation (11) 0x5858  Unknown (22616) <Unknown extend
  31 5.00217700 192.168.1.1 192.168.1.6  ICMP     590 Destination unreachable (Port unreachable)
  34 5.00247300 192.168.1.5 192.168.1.1  DNS      582 Unknown operation (11) 0x5858  Unknown (22616) <Unknown extend
  35 5.00260500 192.168.1.1 192.168.1.5  ICMP     590 Destination unreachable (Port unreachable)
  38 5.00343100 192.168.1.4 192.168.1.1  DNS      582 Unknown operation (11) 0x5858  Unknown (22616) <Unknown extend
  39 5.00351900 192.168.1.1 192.168.1.4  ICMP     590 Destination unreachable (Port unreachable)

⊞ Frame 12: 582 bytes on wire (4656 bits), 582 bytes captured (4656 bits) on interface 0
⊞ Ethernet II, Src: 52:54:00:c5:f0:5f (52:54:00:c5:f0:5f), Dst: 52:54:00:fd:d1:73 (52:54:00:fd:d1:73)
⊞ Internet Protocol Version 4, Src: 192.168.1.5 (192.168.1.5), Dst: 192.168.1.1 (192.168.1.1)
⊞ User Datagram Protocol, Src Port: 60635 (60635), Dst Port: 53 (53)
⊞ Domain Name System (query)
⊞ [Malformed Packet: DNS]

0000  52 54 00 fd d1 73 52 54  00 c5 f0 5f 08 00 45 00   RT...sRT ..._..E.
0010  02 38 f7 16 01 72 40 11  fc d5 c0 a8 01 05 c0 a8   .8...r@. ........
0020  01 01 58 58 58 58 58 58  58 58 58 58 58 58 58 58   ..XXXXXX XXXXXXXX
0030  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0040  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0050  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0060  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0070  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0080  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0090  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00a0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00b0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00c0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00d0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00e0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00f0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0100  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0110  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
```

**Figure 8.** Report from WireShark for UDP traffic


tcp1000-5bota.pcapng [Wireshark 1.12.0 (v1.12.0-0-g4fab41a from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp
Expression... Clear Apply Save

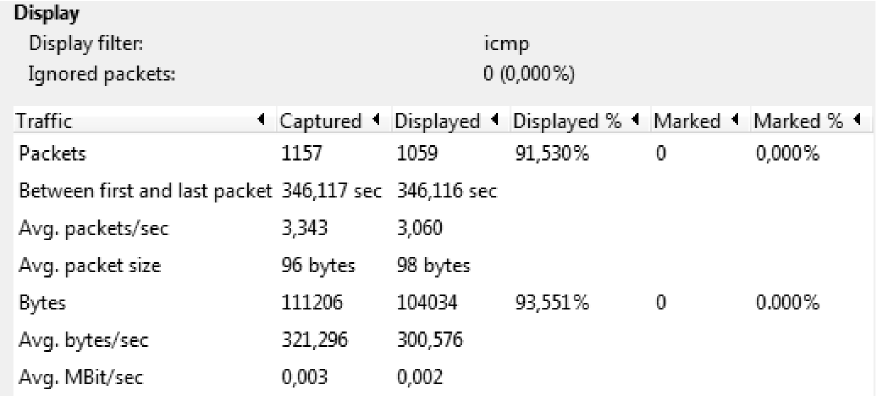
```
No.  Time       Source      Destination  Protocol Length Info
   6 0.00120700 192.168.1.4 192.168.1.1  TCP      74 33938→80 [SYN] Seq=0 Win=4380 Len=0 MSS=1460 SACK_PERM=1 TSval=3419042 TSecr=0 WS=32
   7 0.00121500 192.168.1.6 192.168.1.1  TCP      74 46869→80 [SYN] Seq=0 Win=4380 Len=0 MSS=1460 SACK_PERM=1 TSval=3443211 TSecr=0 WS=32
   8 0.00122000 192.168.1.7 192.168.1.1  TCP      74 53119→80 [SYN] Seq=0 Win=4380 Len=0 MSS=1460 SACK_PERM=1 TSval=966029 TSecr=0 WS=32
   9 0.00132700 192.168.1.5 192.168.1.1  TCP      74 36061→80 [SYN] Seq=0 Win=4380 Len=0 MSS=1460 SACK_PERM=1 TSval=3276081 TSecr=0 WS=32
  14 0.00162200 192.168.1.8 192.168.1.1  TCP      74 57067→80 [SYN] Seq=0 Win=4380 Len=0 MSS=1460 SACK_PERM=1 TSval=547582 TSecr=0 WS=32
  19 0.00176800 192.168.1.1 192.168.1.4  TCP      54 80→33938 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  20 0.00178900 192.168.1.1 192.168.1.5  TCP      54 80→36061 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  21 0.00181100 192.168.1.1 192.168.1.6  TCP      54 80→46869 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  23 0.00184400 192.168.1.1 192.168.1.7  TCP      54 80→53119 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  25 0.00202700 192.168.1.1 192.168.1.8  TCP      54 80→57067 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  26 0.00233700 192.168.1.4 192.168.1.1  TCP      74 33939→80 [SYN] Seq=0 Win=4380 Len=0 MSS=1460 SACK_PERM=1 TSval=3419043 TSecr=0 WS=32
  27 0.00238400 192.168.1.1 192.168.1.4  TCP      54 80→33939 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  28 0.00240900 192.168.1.6 192.168.1.1  TCP      74 46870→80 [SYN] Seq=0 Win=4380 Len=0 MSS=1460 SACK_PERM=1 TSval=3443212 TSecr=0 WS=32
  29 0.00244500 192.168.1.1 192.168.1.6  TCP      54 80→46870 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  30 0.00250000 192.168.1.7 192.168.1.1  TCP      74 53120→80 [SYN] Seq=0 Win=4380 Len=0 MSS=1460 SACK_PERM=1 TSval=966031 TSecr=0 WS=32
  31 0.00250700 192.168.1.5 192.168.1.1  TCP      74 36062→80 [SYN] Seq=0 Win=4380 Len=0 MSS=1460 SACK_PERM=1 TSval=3276082 TSecr=0 WS=32
  32 0.00254000 192.168.1.1 192.168.1.7  TCP      54 80→53120 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  33 0.00256500 192.168.1.1 192.168.1.5  TCP      54 80→36062 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  34 0.00264200 192.168.1.8 192.168.1.1  TCP      74 57068→80 [SYN] Seq=0 Win=4380 Len=0 MSS=1460 SACK_PERM=1 TSval=547583 TSecr=0 WS=32
  35 0.00266800 192.168.1.1 192.168.1.8  TCP      54 80→57068 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

**Figure 9.** Report from WireShark for TCP traffic


ping5.pcapng [Wireshark 1.12.0 (v1.12.0-0-g4fab41a from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: icmp
Expression... Clear Apply Save

```
No.  Time       Source      Destination  Protocol Length Info
   5 0.01580600 192.168.1.4 192.168.1.1  ICMP     98 Echo (ping) request  id=0x071a, seq=1/256, ttl=64 (reply in 8)
   8 0.01646000 192.168.1.1 192.168.1.4  ICMP     98 Echo (ping) reply    id=0x071a, seq=1/256, ttl=128 (request in 5)
   9 0.01888200 192.168.1.5 192.168.1.1  ICMP     98 Echo (ping) request  id=0x0745, seq=1/256, ttl=64 (reply in 12)
  12 0.01944000 192.168.1.1 192.168.1.5  ICMP     98 Echo (ping) reply    id=0x0745, seq=1/256, ttl=128 (request in 9)
  13 0.02055000 192.168.1.6 192.168.1.1  ICMP     98 Echo (ping) request  id=0x06fd, seq=1/256, ttl=64 (reply in 16)
  16 0.02108500 192.168.1.1 192.168.1.6  ICMP     98 Echo (ping) reply    id=0x06fd, seq=1/256, ttl=128 (request in 13)
  17 0.05761300 192.168.1.8 192.168.1.1  ICMP     98 Echo (ping) request  id=0x070a, seq=1/256, ttl=64 (reply in 20)
  20 0.05824600 192.168.1.1 192.168.1.8  ICMP     98 Echo (ping) reply    id=0x070a, seq=1/256, ttl=128 (request in 17)
  21 0.06818800 192.168.1.7 192.168.1.1  ICMP     98 Echo (ping) request  id=0x070a, seq=1/256, ttl=64 (reply in 24)
  24 0.06882900 192.168.1.1 192.168.1.7  ICMP     98 Echo (ping) reply    id=0x070a, seq=1/256, ttl=128 (request in 21)
  25 1.01736600 192.168.1.4 192.168.1.1  ICMP     98 Echo (ping) request  id=0x071a, seq=2/512, ttl=64 (reply in 26)
  26 1.01745900 192.168.1.1 192.168.1.4  ICMP     98 Echo (ping) reply    id=0x071a, seq=2/512, ttl=128 (request in 25)
  27 1.02025800 192.168.1.5 192.168.1.1  ICMP     98 Echo (ping) request  id=0x0745, seq=2/512, ttl=64 (reply in 28)
  28 1.02034200 192.168.1.1 192.168.1.5  ICMP     98 Echo (ping) reply    id=0x0745, seq=2/512, ttl=128 (request in 27)
  29 1.02175200 192.168.1.6 192.168.1.1  ICMP     98 Echo (ping) request  id=0x06fd, seq=2/512, ttl=64 (reply in 30)
  30 1.02178700 192.168.1.1 192.168.1.6  ICMP     98 Echo (ping) reply    id=0x06fd, seq=2/512, ttl=128 (request in 29)
  31 1.05907400 192.168.1.8 192.168.1.1  ICMP     98 Echo (ping) request  id=0x070a, seq=2/512, ttl=64 (reply in 32)
  32 1.05917600 192.168.1.1 192.168.1.8  ICMP     98 Echo (ping) reply    id=0x070a, seq=2/512, ttl=128 (request in 31)
  33 1.06956000 192.168.1.7 192.168.1.1  ICMP     98 Echo (ping) request  id=0x070a, seq=2/512, ttl=64 (reply in 34)
  34 1.06964700 192.168.1.1 192.168.1.7  ICMP     98 Echo (ping) reply    id=0x070a, seq=2/512, ttl=128 (request in 33)
```

**Figure 10.** Report from WireShark for ICMP traffic



**Display**

Display filter: icmp

Ignored packets: 0 (0,000%)

| Traffic | Captured | Displayed | Displayed % | Marked | Marked % |
|---|---|---|---|---|---|
| Packets | 1157 | 1059 | 91,530% | 0 | 0,000% |
| Between first and last packet | 346,117 sec | 346,116 sec | | | |
| Avg. packets/sec | 3,343 | 3,060 | | | |
| Avg. packet size | 96 bytes | 98 bytes | | | |
| Bytes | 111206 | 104034 | 93,551% | 0 | 0,000% |
| Avg. bytes/sec | 321,296 | 300,576 | | | |
| Avg. MBit/sec | 0,003 | 0,002 | | | |

**Figure 11.** General report from WireShark

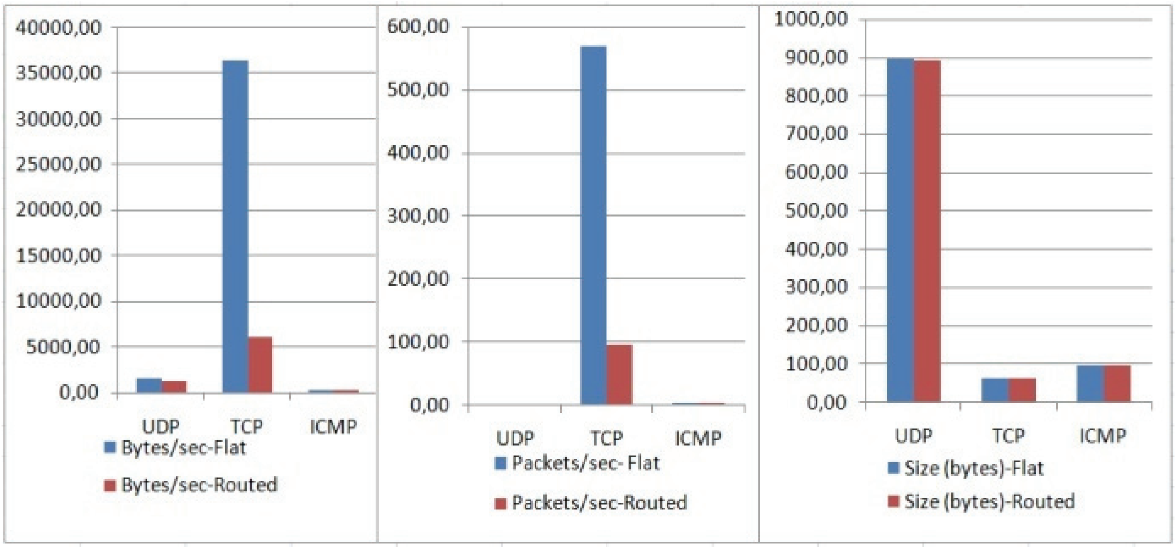


**Figure 12.** Comparison of the results

```
IPTABLES = "/usr/sbin/iptables"
/sbin/modprobe ip_tables
/sbin/modprobe iptable_filter
/sbin/modprobe ipt_state
$IPTABLES -F INPUT
$IPTABLES -F OUTPUT
$IPTABLES -F FORWARD
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P OUTPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT
$IPTABLES -A INPUT -p tcp --sport 6667 -j DROP
$IPTABLES -A OUTPUT -p tcp --dport 6667 -j DROP
```

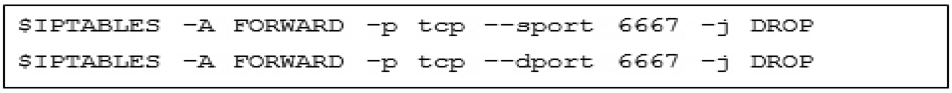
**Figure 13.** Iptables rules

lower. This is due to the fact that in the routed network the traffic passes through different networks until it reaches the destination – the attacked machine. For TCP attack, however the differences are significant, which is due to the specific mechanism of the protocol for acknowledgment and retransmission. The graphical representation of the results is shown on *figure 12.*

For ICMP attack is typical that the bots can not complete their attack, as the attacked machine can not send *Echo Reply* within the specified to bot machines *ping* timeout. This is proof that the attack achieved its goal before it reaches the maximum value of parameters set in the command.

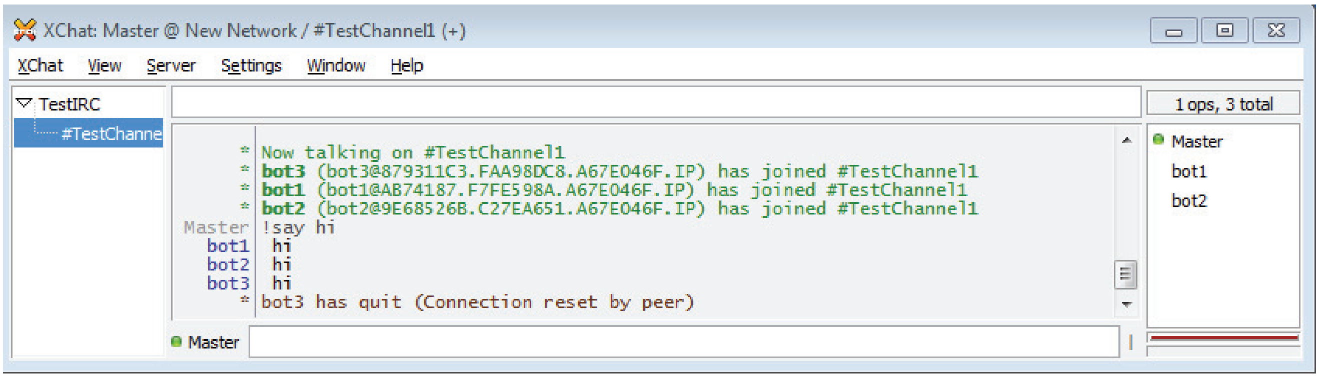
## A Possible Solution for Prevention

A possible solution for prevention from bot infection is to use the Linux Netfilter framework which allows controlling of packet traffic. The current implementation of the framework is the *iptables* library. To verify the solution the protection is tested on routed network shown on *figure 5.* On routers PC1 and PC3 are written iptables rules, which



```
$IPTABLES -A FORWARD -p tcp --sport 6667 -j DROP
$IPTABLES -A FORWARD -p tcp --dport 6667 -j DROP
```

**Figure 14.** Adding new rules



```
       * Now talking on #TestChannel1
       * bot3 (bot3@879311C3.FAA98DC8.A67E046F.IP) has joined #TestChannel1
       * bot1 (bot1@AB74187.F7FE598A.A67E046F.IP) has joined #TestChannel1
       * bot2 (bot2@9E68526B.C27EA651.A67E046F.IP) has joined #TestChannel1
  Master | !say hi
    bot1 | hi
    bot2 | hi
    bot3 | hi
       * bot3 has quit (Connection reset by peer)
```

**Figure 15.** Blocking of a bot

purpose is to restrict the bot 3 (PC2) to access the network. On *figure 13* the basic iptables configuration for both the routers is shown.

After activating the rules, the bots were started. The bot 3 connects to the network and runs the command **!say hi**. While the bots are active, new iptables rules were added on routers 0 (*figure 14*).

As the result the bots are still active in the channel, but after running the rules, the connection with bot 3 is suspended (*figure 15*).

## Conclusion

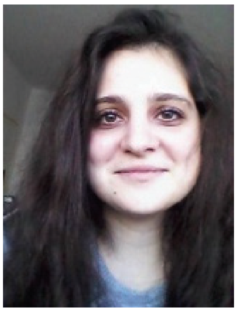
In this paper are presented some aspects of the implementation of the simulation environment for generating IRC DoS network attacks. The simulator provides flexible tools with which can be simulated different types of attacks with various parameters both in real conditions and in a laboratory. A possible solution for bot attacks preventing is presented.

For future work is planned expansion with generators of other network attacks, and adding opportunities for realization of DDoS attacks. Another aspect is the use of threads to enhance the effectiveness of the simulator.

## References

1. Elisan, C. C. Malware, Rootkits & Botnets A beginner's Guide. McGraw-Hill Education – Europe, ISBN 9780071792066, 2012, 55-82.
2. LOIC. https://sourceforge.net/projects/loic.
3. Shankdhar, P. DoS Attacks and Free DoS Attacking Tools. INFOSEC Institute, 2013.
4. MazeBolt Security. https://mazebolt.com/.
5. Bonesi. The DDoS Botnet Simulator. https://code.google.com/p/bonesi.
6. Waldecker, B. A Review on IRC Botnet Detection and Defence. St. Poelten University of Applied Sciences, 2013, 1-9.
7. Belaton, B., A. H. R. Awadi. Multi-phase IRC Botnet and Botnet Behavior Detection Model. – *International Journal of Computer Applications,* 66, 2013, No. 15, 2-10.

**Manuscript received on 28.11.2016**



***Yuliya Aleksieva*** *received her BsC of Software and Internet Technologies from Technical University of Varna, Bulgaria in 2016. She is currently MsC of Software Engineering at the Department of Computer Science, Technical University of Varna, Bulgaria. Since 2015 she works as embedded software engineer at ASIC Depot Ltd. Her current research interests include botnets, network attacks and intrusion-detection systems.*

***Hristo Valchanov*** *is currently Associate Professor at Technical University of Varna, Bulgaria. He is the Head of Computer Science and Engineering Department. His research interests include network security, operating systems, parallel and distributed computing and distributed simulation.*

*Contacts:*
*Technical University of Varna*
*Department of Computer Science*
*e-mail: hristo@tu-varna.bg*