International Journal of Computer Applications Technology and Research
Volume 5–Issue 12, 733-747, 2016, ISSN:-2319–8656

Empirical Investigation of Instant Messaging Security in a Virtual Environment

Peter S. Nyakomitta
School of informatics and Innovative systems
Jaramogi Oginga Odinga University of Science and Technology
Kenya

Dr. Solomon Ogara
School of informatics and Innovative systems
Jaramogi Oginga Odinga University of Science and Technology
Kenya

Dr. Silvance Abeka
School of informatics and Innovative systems
Jaramogi Oginga Odinga University of Science and Technology
Kenya

Abstract **-** Use of instant messaging services is becoming increasingly popular with Internet based systems like America Online's Instant Messaging (AIM), Microsoft's MSN Messenger, Yahoo! Messenger, WhatsApp, Viber, Kakaotalk, Skype and face book instant messenger. These tools support any process where quick response and rapid problem solving are needed, and where faster communication than emails or telephones is useful. More and more people are enjoying the convenience and simplicity provided by the real-time messaging systems in their day-to-day life. Moreover, the instant messaging services have also found applications in business. In this application domain, the instant messaging services are employed for communicating with customers and partners, offering customer support, receiving real-time alerts, as well as management and project coordination. Despite their heavy utilization, public instant messaging systems have been criticized for having a number of security weaknesses. These weaknesses originate from the facts that the instant messaging clients are always on, those logs can contain sensitive information, and that the communication goes through an externally controlled server. Most of the instant messaging services were never intended for secure communication in the first place. The rapid growth in the number of public instant messaging users has therefore created a new security concern for information technology managers. In this paper, a prototype instant messaging was developed and employed to investigate some of the security challenges in instant messaging applications. The results indicated that upon following the TCP stream, the instant messages were in plain text in the sending and receiving communication devices interfaces and therefore prone to eavesdropping. As such, the researchers propose a port-based algorithm that would scramble the data packets at the end devices, requiring the users to input decryption keys for the data to be transformed into human readable format.

Keywords: ***Instant Messaging (IM), Plain Text, prototype, Client, Server, Security***

1. INTRODUCTION

The potential threats when using instant messaging services include the spread of malicious code, instant messaging software vulnerabilities, leakage of sensitive information, monitoring and retention issues, and lack of accountability. In their study, Weissbrot and Alison (2016) explain that the enterprise usage of instant is growing in both volume and importance. The use of these tools benefit their users in facilitating faster decision-making process, higher productivity and lower telecommunication costs. At the same time as, the instant messaging threats such as viruses are rapidly gaining attention as attackers begin to shift their focus from better-protected email systems to these networks.

Moreover, Mark (2015) report that spam messages can also be spread through the instant messaging tools. The spam that a user receives via these services is referred to as spim. Popular instant messaging clients have, just like any other software application, have a history of common security vulnerabilities. This means that installing an instant messaging client has the potential of introducing new vulnerabilities to a computer system. Confidentiality, which deals with the protection of organizational data or government data from illegal access, is a major concern when using a public instant messaging service for communication. This so according to Vinnie and Belvin, (2012), because in public instant messaging networks, communications exchanged between users are normally routed through instant messaging server farms which are controlled by the service providers themselves. In situations where client instant messaging software has a peer-to-peer capability, users can communicate with each other without passing through these servers.

Lin et al., (2016) note that no matter which mode is being utilized for communication purposes, this traffic is vulnerable to eavesdropping because most public clients do not possess any encryption capability. The consequences are that it is possible for sensitive information to be read or sniffed by unauthorized users. The situation can be even worse when public instant messaging services are used to communicate with individuals outside an organization. This may lead to the leakage of sensitive organizational classified data.

11. LITERATURE REVIEW

In this section, the researcher discuss on the literature concerning the related work to the research, which is empirical investigation of instant messaging security in a virtual environment.

2.1. RELATED WORK

In his research work, Wendell (2013) found out that the protocols employed by public instant messaging services are often considered rogue protocols. This because they were specifically designed to evade standard security controls. The consequences are that not only can instant messaging clients be configured to connect through SOCKS or web proxy servers, but as Green (2014) point out, the protocols are also capable of finding their way out through the firewall on their own. They can do this by determining an open port such as transmission control protocol (TCP) port 80, or by tunneling their traffic inside the hypertext transfer protocol (HTTP) requests. These practices make these traffic unrecognizable from standard web traffic. Additionally, the scripting and file transfer capabilities of instant messaging systems might expose an organization to leaks of sensitive information.

According to Andreas and Buchenscheit (2014), most of the productive features provided by the instant messaging applications are only one side of the coin. This is because these applications have vulnerabilities that related to the underlying instant messaging technology. Consequently, these vulnerabilities expose user communications to a number of security threats. Therefore, communication via these applications is regarded insecure. To start with, Jagwani (2016) note that all of the messages and connection information are retained on the application providers’ servers. This means that the information communicated across the network is controlled by the provider of the instant messaging utility.

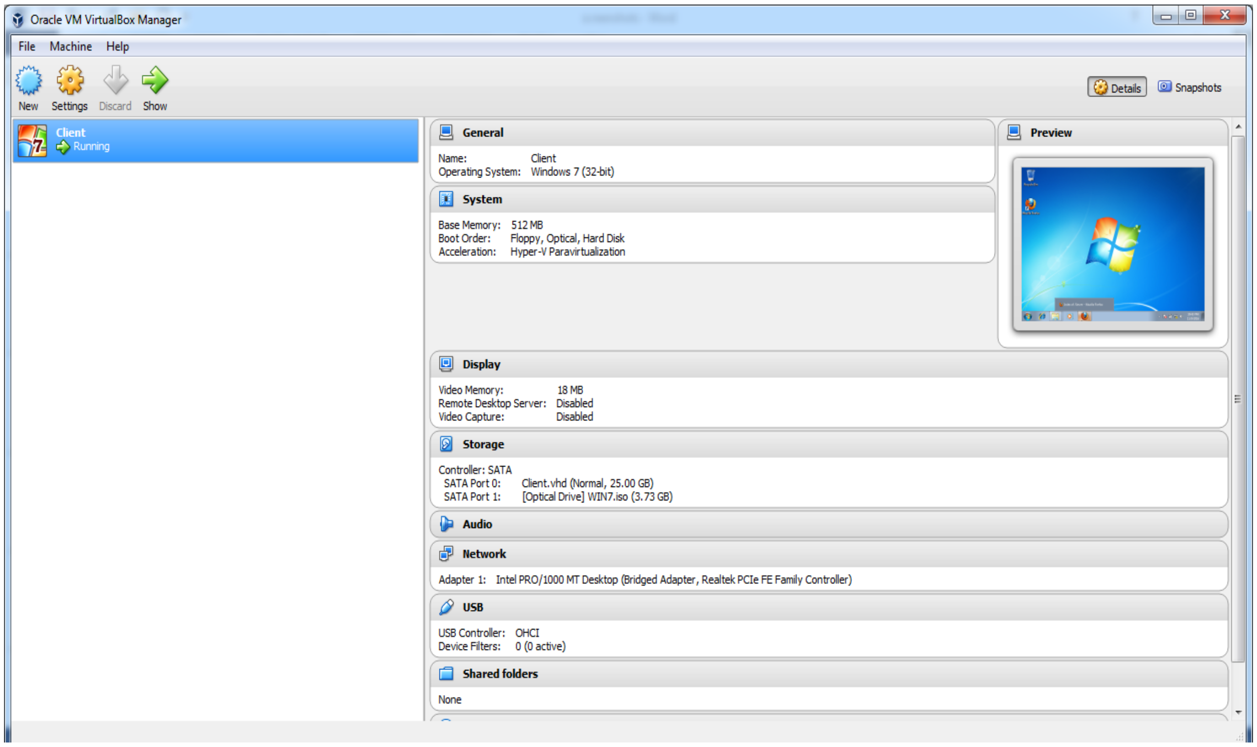
Another serious challenge with instant messaging connections is that the messaging process normally happens in plain text (Frosch et al., 2016). Consequently, this renders them susceptible to eavesdropping. Additionally, instant messaging client software quite often requires the user to expose their open user datagram ports. This gives rise to the threat posed by potential security vulnerabilities of user datagram protocol (UDP) ports. As Smith (2013) points out, UDP is vulnerable to spoofing and denial of service attacks. On the other hand, it is not feasible to spoof an address across the internet using transmission control protocol (TCP). This is because the three way handshake will never complete. Moreover, there are features of instant messaging applications that are threats to security. Such features include presence and status broadcasting, interoperability with others, maintaining a lists of all desired contacts, use of third party servers to provide chat functionality to messenger clients and keeping a log of messages and other events/ activities (Fahrnberger, 2014).

111. RESEARCH METHODOLOGY

This section presents the way the study was carried out. It involves the various steps that the researcher adopted in studying the research problem as well the logic behind those steps which include the research prototyping approach and the procedure is provided as outlined below.

3.1 Prototyping Approach

In this paper, a model that could help demonstrate the security challenges in instant messaging applications in a virtualized environment was developed. The hypervisor was chosen to be Oracle VM Virtual box. This hypervisor is a cross-platform virtualization application, meaning that it installs on the existing Intel or AMD-based computers,. It supports operating systems such as Windows, Mac, Linux or Solaris. It serves to extend the capabilities of the existing computer so that it can run multiple operating systems inside multiple virtual machines at the same time. Using this hypervisor, one can install and run as many virtual machines as he likes. The only practical limiting factors are disk space and memory. Moreover, it is very simple to use and very powerful. This is because it can run on any platform, ranging from small embedded systems or desktop class machines to datacenter deployments and even Cloud environments. This made it the best choice for this paper. Figure 1 shows the interface for VirtualBox hypervisor.



General
Name: Client
Operating System: Windows 7 (32-bit)

System
Base Memory: 512 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: Hyper-V Paravirtualization

Display
Video Memory: 18 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage
Controller: SATA
SATA Port 0: Client.vhd (Normal, 25.00 GB)
SATA Port 1: [Optical Drive] WIN7.iso (3.73 GB)

Audio

Network
Adapter 1: Intel PRO/1000 MT Desktop (Bridged Adapter, Realtek PCIe FE Family Controller)

USB
USB Controller: OHCI
Device Filters: 0 (0 active)

Shared folders
None

Figure 1: VirtualBox Interface

3.2 Procedure

After the installation of the VirtualBox hypervisor, a client operating system was installed inside this hypervisor. This client was chosen to be Windows Professional, 32 bit.

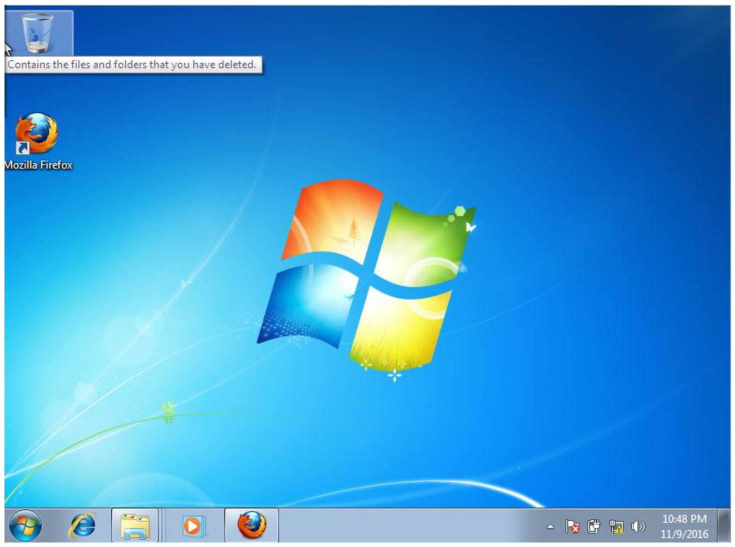


Figure 2: Client Operating System

The 32 – bit client was selected to avoid bus speed mismatch because the underlying host operating system ran 32 –bit. After this, Wamp server was installed inside this client operating system. This was meant to provide a respository for the instant messages through its *PhPMyadmin* feature as shown in Figure 3.

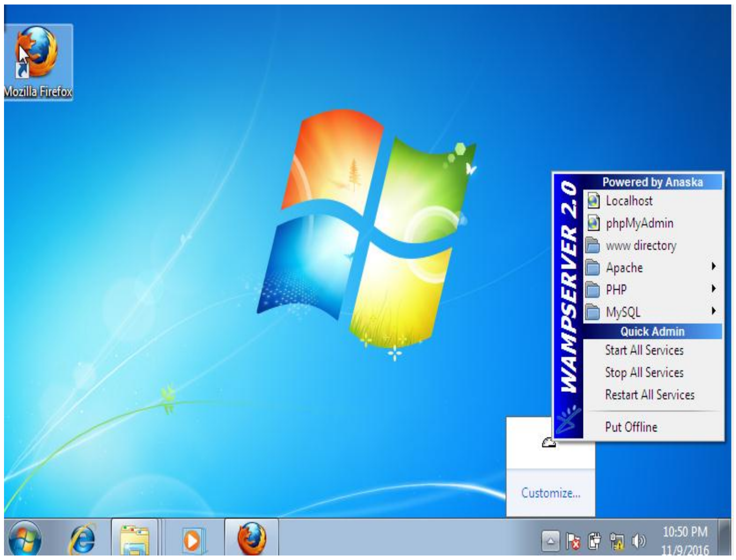


Figure 3: Wamp Server Installation

All the *PhP* coding was done and saved in the *www* folder of *wamp* server. The same *wamp* server was also installed on another machine that was required to achieve a two way instant messaging exchange. An Ethernet cable was then connected between the machines that required exchanging the instant messages and appropriate internet protocol (IP) addresses were assigned using class C subnet. Figure 4 show the typical set up that was employed.

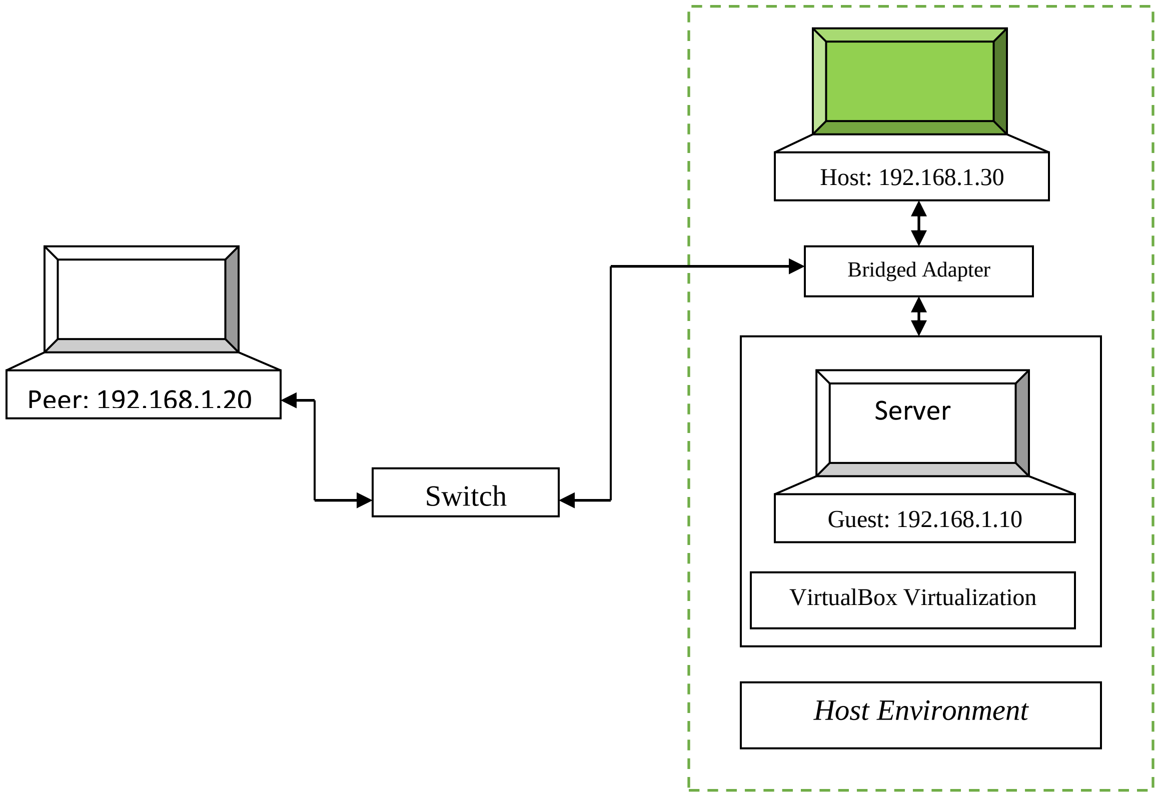


Figure 4: Prototyping Setup

In this paper, the Host operating system refers to the operating system of the physical computer on which VirtualBox was installed. On the other hand, the guest operating system was the operating system that is running inside the virtual machine. Virtual machine (VM) refers to the special environment that VirtualBox creates for the guest operating system while it is running. In other words, the guest operating system is run in a virtual machine. Basically, a virtual machine could be shown as a window on the computer's desktop, but depending on which of the various frontends of VirtualBox is in use, it can be displayed in full screen mode or remotely on another computer.

Technically, VirtualBox regards a virtual machine as a set of parameters that determine its behavior. They include hardware settings (how much memory the virtual machine should have, what hard disks VirtualBox should virtualize through which container files, or what CDs are mounted) as well as state information (whether the virtual machine is currently running, saved, or its snapshots).

Figure 4 shows that the hypervisor was hosted in address *192.168.1.30*. This hypervisor in turn hosted a client of IP address *192.168.1.10*. In order for the guest operating system to communicate with the host operating system, a bridged adapter was employed as shown in Figure 4. In bridged networking, VirtualBox utilizes a device driver on the host system that filters data from the physical network adapter.

For this reason, this driver is called a net filter driver.
This permits VirtualBox to capture data from the physical network and inject data into it. The effect of this is the creation of a new network interface in software. Ideally, when the guest operating system is utilizing such a new software interface, it looks to the host system as though the guest were physically connected to the interface using a network cable. Effectively, the host can send data to the guest through that interface and receive data from it. This means that one can set up routing or bridging between the guest and the rest of your network.

To enable bridged networking, the *Settings* dialog of a virtual machine was opened; from there navigation was done to the *Network* page. Finally, the selection of *Bridged network* was accomplished in the drop down list for the *Attached to* field as shown in Figure 5.

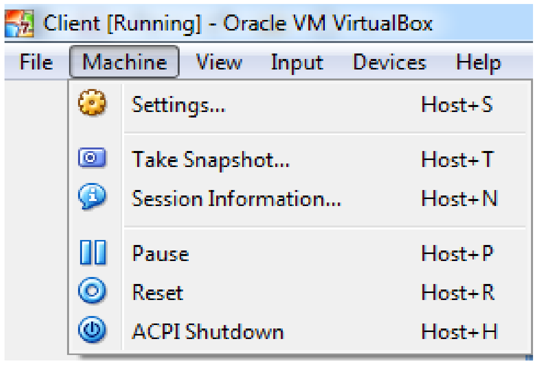
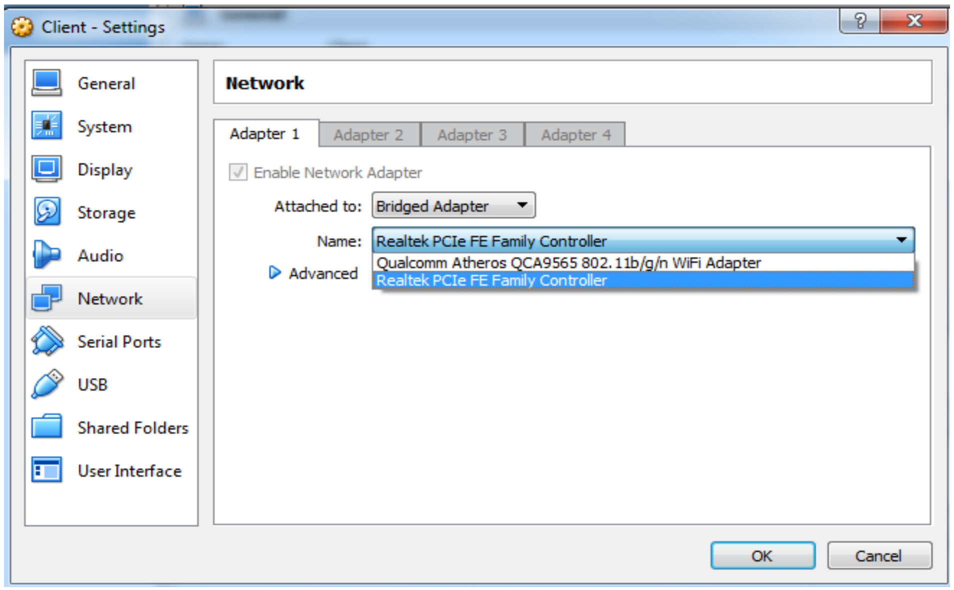


Figure 5: VirtualBox Network Setting

To finish the configuration, the desired host interface was selected from the list at the bottom of the page, which contains the physical network interfaces of the systems. As Figure 6 demonstrates, two network interfaces were detected: *RealTek PCIe FE Family Controller* and *Qualcom Atheros QCA9565802.11b/n Wifi Adapter*. Howver, since the latter adapter was for the wireless connections, the former interface *RealTek PCIe FE Family Controller* was selected.



Network

Enable Network Adapter

Attached to: Bridged Adapter

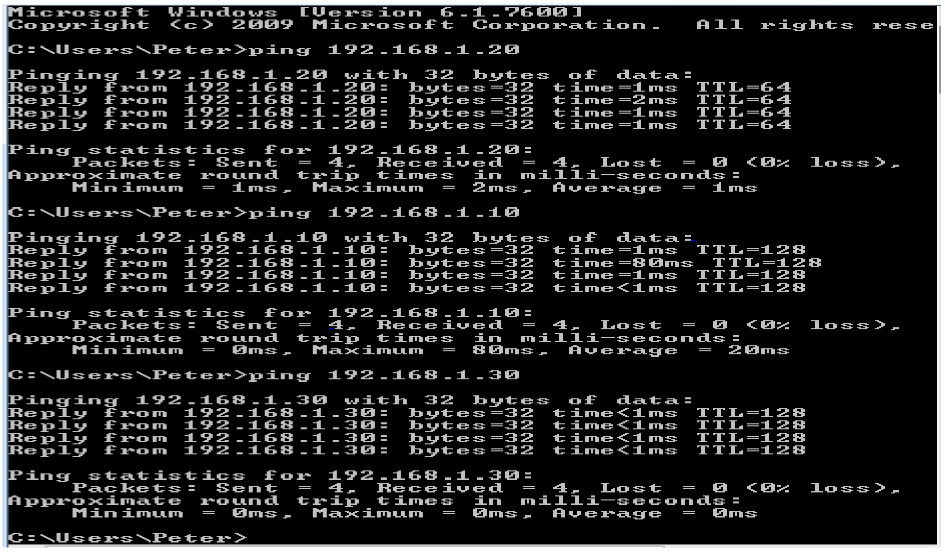
Name: Realtek PCIe FE Family Controller

Qualcomm Atheros QCA9565 802.11b/g/n WiFi Adapter

Realtek PCIe FE Family Controller

Figure 6: Bridged Host-Client Connections

To test the effectiveness of these configurations, the internet control message protocol (ICMP) code number 8, called packet internet groper (PING) utility was employed. Figure 7 shows the results of the PING utility against IP address of the guest running in Oracle VirtualBox VM.



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights rese

C:\Users\Peter>ping 192.168.1.20

Pinging 192.168.1.20 with 32 bytes of data:
Reply from 192.168.1.20: bytes=32 time=1ms TTL=64
Reply from 192.168.1.20: bytes=32 time=2ms TTL=64
Reply from 192.168.1.20: bytes=32 time=1ms TTL=64
Reply from 192.168.1.20: bytes=32 time=1ms TTL=64

Ping statistics for 192.168.1.20:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 2ms, Average = 1ms

C:\Users\Peter>ping 192.168.1.10

Pinging 192.168.1.10 with 32 bytes of data:
Reply from 192.168.1.10: bytes=32 time=1ms TTL=128
Reply from 192.168.1.10: bytes=32 time=80ms TTL=128
Reply from 192.168.1.10: bytes=32 time=1ms TTL=128
Reply from 192.168.1.10: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 80ms, Average = 20ms

C:\Users\Peter>ping 192.168.1.30

Pinging 192.168.1.30 with 32 bytes of data:
Reply from 192.168.1.30: bytes=32 time<1ms TTL=128
Reply from 192.168.1.30: bytes=32 time<1ms TTL=128
Reply from 192.168.1.30: bytes=32 time<1ms TTL=128
Reply from 192.168.1.30: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.30:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Peter>
```

Figure 7: Connectivity Testing

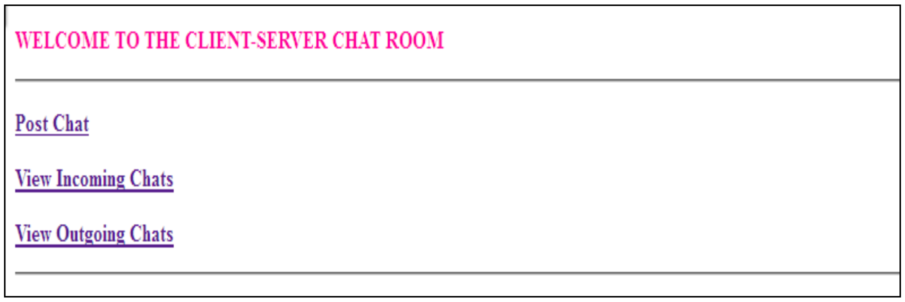
As this figure shows, there was successful communication from the peer instant messaging machine to the virtualized instant messaging guest peer. This meant that all was set for the actual instant message communication between the virtualized guest and the non-virtualized peer.

1V. RESULTS AND DISCUSSION

This section presents and discusses the results from the empirical investigation of Instant Messaging security in a virtualized environment as explained below.

4.1 Client-Server Interface

Each of the instant message communicating entity had the hyperlinks shown in Figure 8. This consisted of the *post Chat*, *View Incoming Chat* and *View Outgoing Chat* hyperlinks. The *Post Chat* link was utilized to send an instant message to the receiver while the *View Incoming Chat* was employed to retrieve the instant message directed towards the user. Finally, the *View Outgoing Chat* link was used to retrieve the instant messages that the user has sent.



WELCOME TO THE CLIENT-SERVER CHAT ROOM

Post Chat

View Incoming Chats

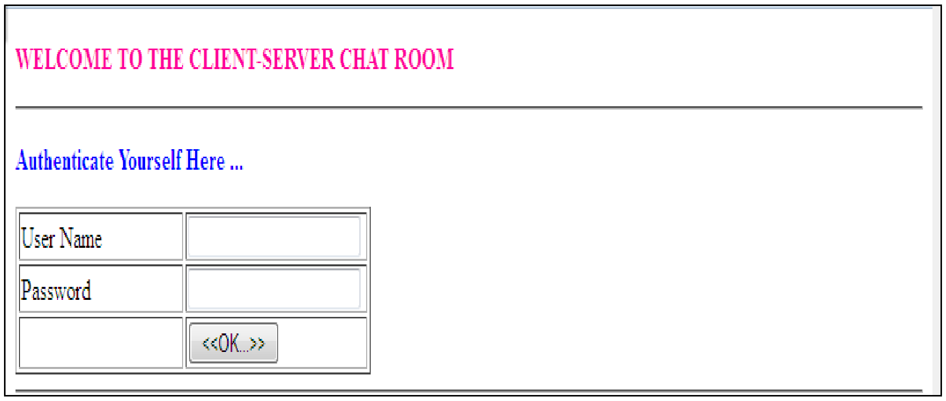
View Outgoing Chats

Figure 8: Main Chat Interface

To post a chat, one clicks the *'Post Chat'* link that then load the interface shown in Figure 9. On this interface, the user authenticates himself by entering a valid user name and password. Afterwards, he clicks the *'OK'* button to load the next interface.

4.2. Authentication Interface

The design represent an interface used by the. users to authenticate themselves when they log into the system.


WELCOME TO THE CLIENT-SERVER CHAT ROOM

Authenticate Yourself Here ...

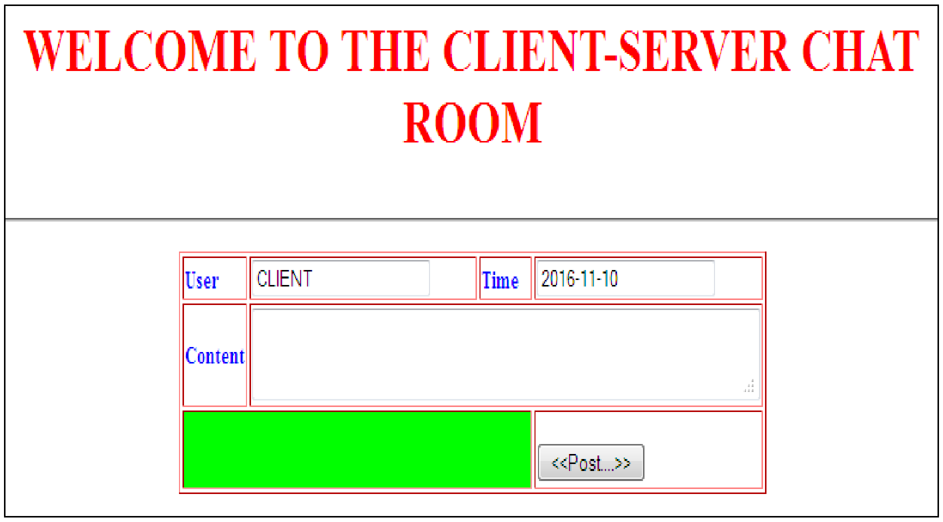
User Name	
Password	
	<<OK...>>

Figure 9: Authentication Interface

The username and password are analogous to the password or the unique pattern that the user draws to unlock the phone so as to access the instant messaging interface. Upon the entry of the correct credentials, the user is presented with the interface shown in Figure 10.

4.3. Composing Interface

In order to compose a message that will be instantly posted from the peers to the server for processing, figure 10 represents the design.


WELCOME TO THE CLIENT-SERVER CHAT ROOM

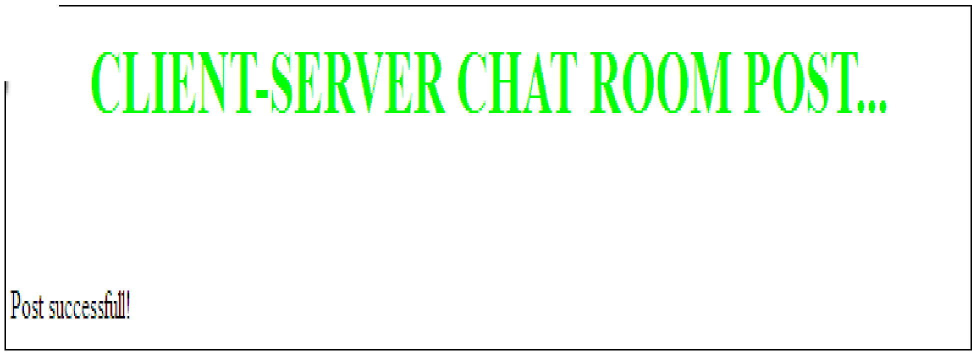
User	CLIENT	Time	2016-11-10
Content			
🟩		<<Post...>>	

Figure 10: Chat Posting Interface

In this interface, the user types his instant chat message and clicks the *'OK'* command button. This will in turn load the interface shown in Figure 11.

4.4. Posting Response

This serves to inform the user that indeed the chat he posted has been delivered to the recipient, otherwise an error message will be displayed. To retrieve the incoming instant messages, one simply clicks on *"View Incoming Chats"* link of Figure 8. Upon doing this, the interface on Figure 12 will be displayed.



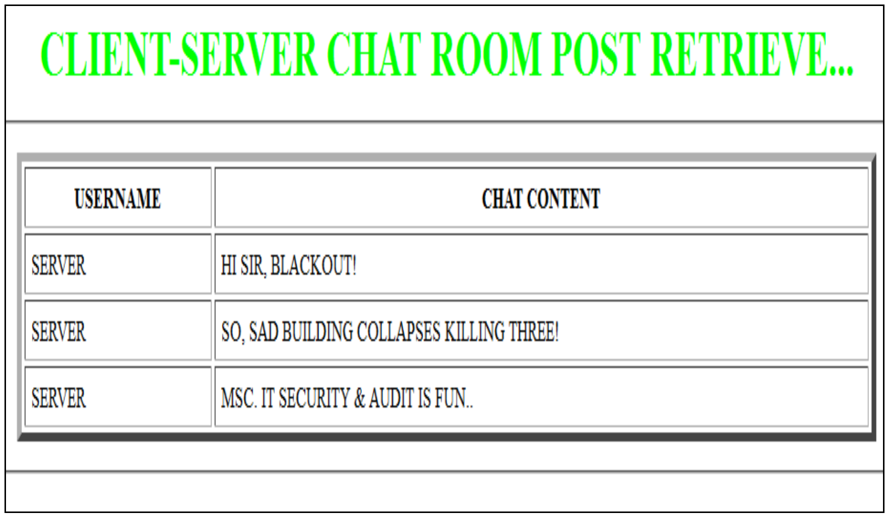
CLIENT-SERVER CHAT ROOM POST...

Post successfull!

Figure 11: Instant Message Posting Successful

4.5. Incoming Message

This process clearly depicts what happens with normal instant chat message retrieval applications.



CLIENT-SERVER CHAT ROOM POST RETRIEVE...

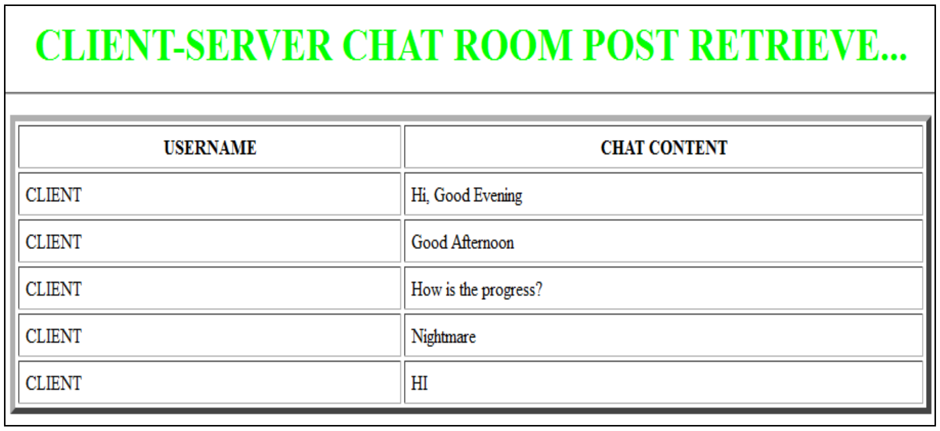
USERNAME	CHAT CONTENT
SERVER	HI SIR, BLACKOUT!
SERVER	SO, SAD BUILDING COLLAPSES KILLING THREE!
SERVER	MSC. IT SECURITY & AUDIT IS FUN..

Figure 12: Incoming Chats

Typically, one clicks on the sender's contact, which then loads the chat interface from which the incoming chats can be read and post messages. To retrieve the chats sent, one clicks the *"View Outgoing Chats"* and this displays the information shown in Figure 13.

4.6. Outgoing Message

The design demonstrate the content of the outgoing messages. On both Figure12 and Figure 13, the name of the sender and receiver are displayed on the *"Username"* column. The actual chat is displayed under the *"Chat Content"* column.



CLIENT-SERVER CHAT ROOM POST RETRIEVE...

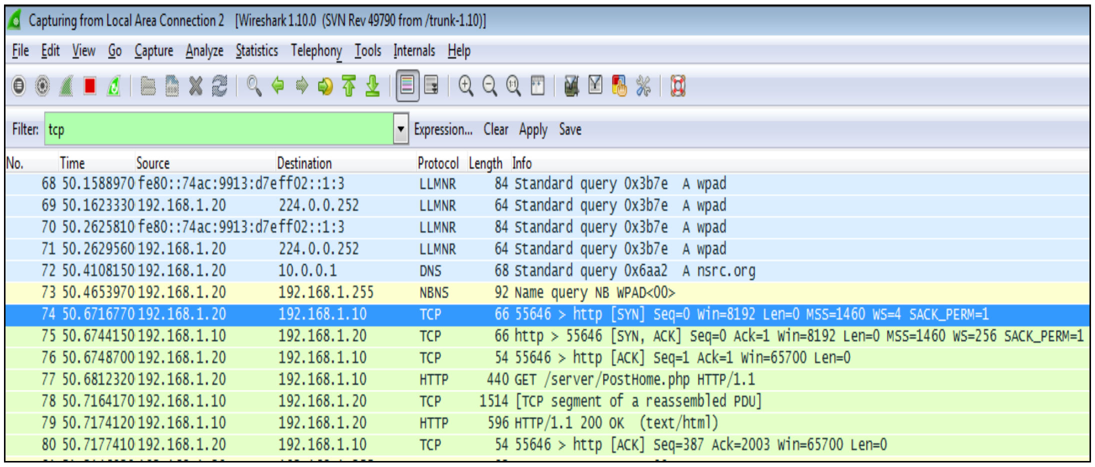
USERNAME	CHAT CONTENT
CLIENT	Hi, Good Evening
CLIENT	Good Afternoon
CLIENT	How is the progress?
CLIENT	Nightmare
CLIENT	HI

Figure 13: Outgoing Chat Retrieval

Clearly, these instant message communications are in plain text, raising the susceptibility to eavesdropping. Further traffic analysis using the Wireshark software was conducted to determine whether the instant message communication can be intercepted. The results obtained are as shown in Figure 14.

4.7 TCP Three Way Handshake communication

The figure illustrates the synchronization process of the device during communication which describes a Three-way handshake. The TCP allows one side to establish a connection as a client and the other side to accept the connection or reject it as a serve. It shows that there was a TCP communication between the peer (IP address 192.168.1.20) and the virtualized guest (IP address 192.168.1.10).


Capturing from Local Area Connection 2 [Wireshark 1.10.0 (SVN Rev 49790 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp
Expression... Clear Apply Save

```
No.  Time        Source                      Destination     Protocol Length Info
  68 50.1588970  fe80::74ac:9913:d7eff02::1:3                LLMNR      84 Standard query 0x3b7e  A wpad
  69 50.1623330  192.168.1.20               224.0.0.252     LLMNR      64 Standard query 0x3b7e  A wpad
  70 50.2625810  fe80::74ac:9913:d7eff02::1:3                LLMNR      84 Standard query 0x3b7e  A wpad
  71 50.2629560  192.168.1.20               224.0.0.252     LLMNR      64 Standard query 0x3b7e  A wpad
  72 50.4108150  192.168.1.20               10.0.0.1        DNS        68 Standard query 0x6aa2  A nsrc.org
  73 50.4653970  192.168.1.20               192.168.1.255   NBNS       92 Name query NB WPAD<00>
  74 50.6716770  192.168.1.20               192.168.1.10    TCP        66 55646 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
  75 50.6744150  192.168.1.10               192.168.1.20    TCP        66 http > 55646 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
  76 50.6748700  192.168.1.20               192.168.1.10    TCP        54 55646 > http [ACK] Seq=1 Ack=1 Win=65700 Len=0
  77 50.6812320  192.168.1.20               192.168.1.10    HTTP      440 GET /server/PostHome.php HTTP/1.1
  78 50.7164170  192.168.1.10               192.168.1.20    TCP      1514 [TCP segment of a reassembled PDU]
  79 50.7174120  192.168.1.10               192.168.1.20    HTTP      596 HTTP/1.1 200 OK  (text/html)
  80 50.7177410  192.168.1.20               192.168.1.10    TCP        54 55646 > http [ACK] Seq=387 Ack=2003 Win=65700 Len=0
```

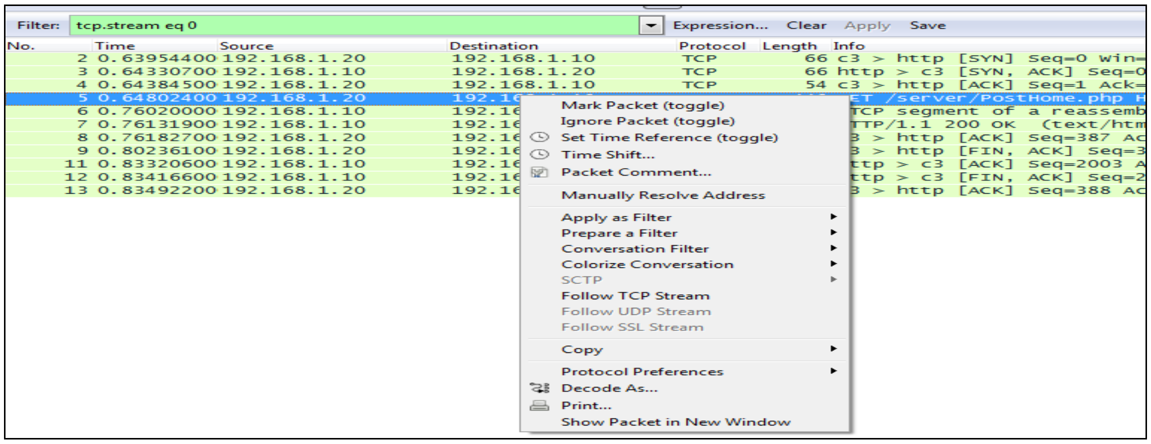
Figure 14: Instant Message Communication Interception

The sequence number of the initial packet was 0. The guest then respondent by acknowledging the request for connection, with ACK=1. The peer and the virtualized guest then proceeded to exchange data using the hypertext transfer protocol (HTTP).

4.8 Transmission Control Protocol Stream

To demonstrate that the instant messages were in plain text in both sending and transmitting devices, the TCP stream was followed as shown in Figure 15.

It's a window that details the "request" sent and the "response" received. This process was done for both the server and the client. The idea was to capture the live packet as it was being transmitted from the client to the server and from the server to the client. Since the address of the client was 192.168.1.20 and that of the server was 192.168.1.10, the following communication shown in Figure 16 was intercepted and live capture carried out.

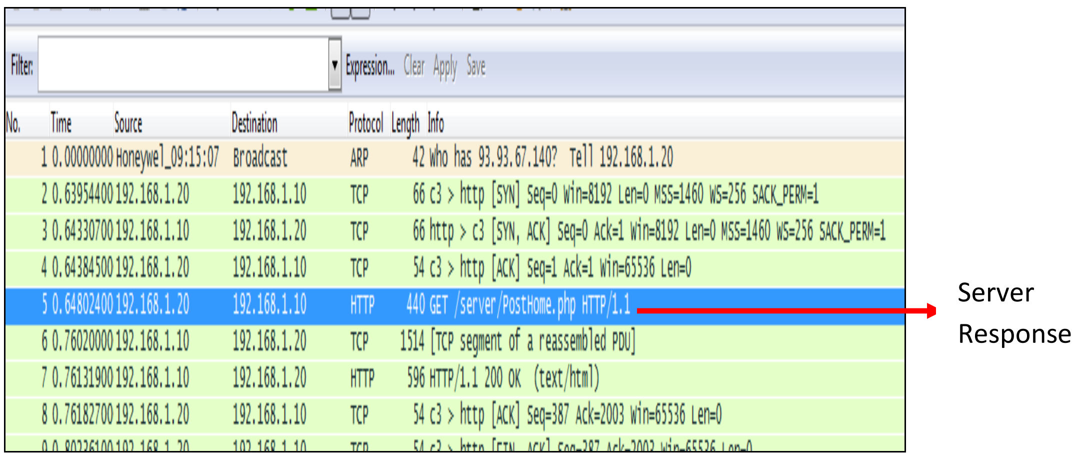


```
Filter: tcp.stream eq 0
No.   Time        Source         Destination    Protocol Length Info
   2 0.63954400 192.168.1.20   192.168.1.10   TCP        66 c3 > http [SYN] Seq=0 Win=
   3 0.64330700 192.168.1.10   192.168.1.20   TCP        66 http > c3 [SYN, ACK] Seq=0
   4 0.64384500 192.168.1.20   192.168.1.10   TCP        54 c3 > http [ACK] Seq=1 Ack=
   5 0.64802400 192.168.1.20   192.16                      ET /server/PostHome.php H
   6 0.76020000 192.168.1.10   192.16                      TCP segment of a reassemb
   7 0.76131900 192.168.1.10   192.16                      TTP/1.1 200 OK  (text/htm
   8 0.76182700 192.168.1.20   192.16                      3 > http [ACK] Seq=387 Ac
   9 0.80236100 192.168.1.20   192.16                      3 > http [FIN, ACK] Seq=3
  11 0.83320600 192.168.1.10   192.16                      ttp > c3 [ACK] Seq=2003 A
  12 0.83416600 192.168.1.10   192.16                      ttp > c3 [FIN, ACK] Seq=2
  13 0.83492200 192.168.1.20   192.16                      3 > http [ACK] Seq=388 Ac
```

Figure 15: TCP Stream Following

4.9 HTTP Client-Sarver Communication

The figure depict response packet capture from the server through hypertext transfer protocol.



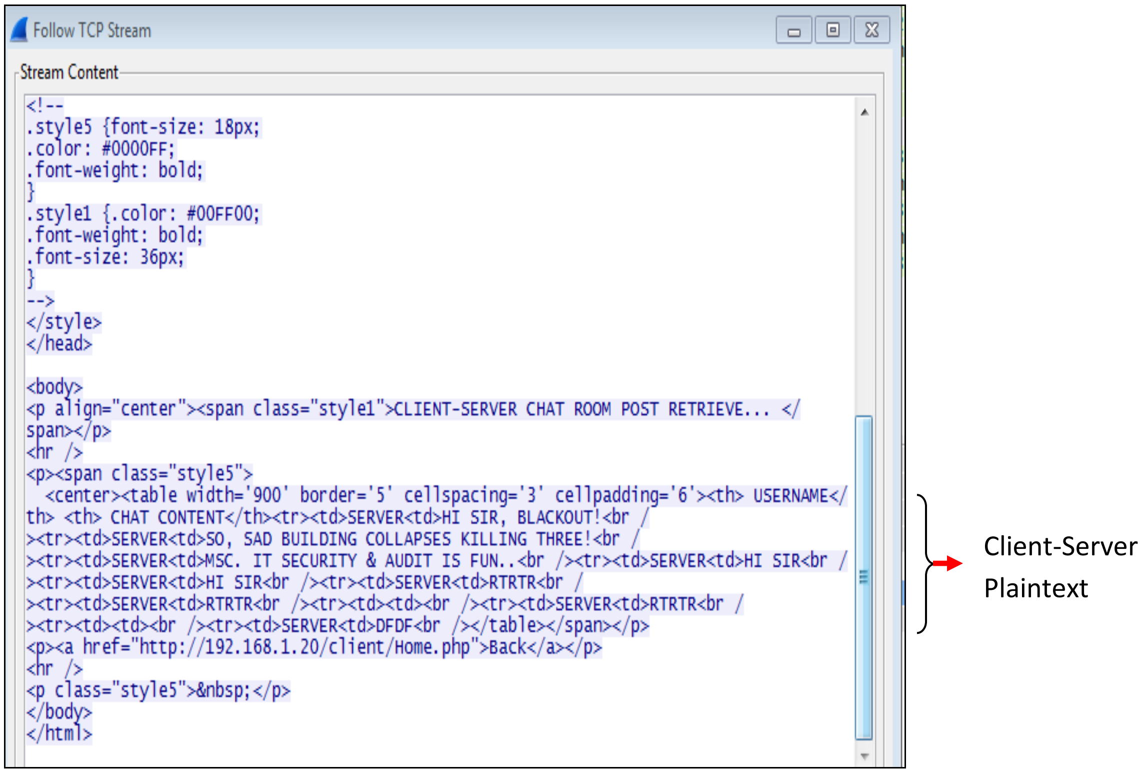
```
No. Time       Source              Destination    Protocol Length Info
  1 0.00000000 Honeywel_09:15:07   Broadcast      ARP     42 Who has 93.93.67.140?  Tell 192.168.1.20
  2 0.63954400 192.168.1.20        192.168.1.10   TCP     66 c3 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
  3 0.64330700 192.168.1.10        192.168.1.20   TCP     66 http > c3 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
  4 0.64384500 192.168.1.20        192.168.1.10   TCP     54 c3 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
  5 0.64802400 192.168.1.20        192.168.1.10   HTTP   440 GET /server/PostHome.php HTTP/1.1
  6 0.76020000 192.168.1.10        192.168.1.20   TCP   1514 [TCP segment of a reassembled PDU]
  7 0.76131900 192.168.1.10        192.168.1.20   HTTP   596 HTTP/1.1 200 OK  (text/html)
  8 0.76182700 192.168.1.20        192.168.1.10   TCP     54 c3 > http [ACK] Seq=387 Ack=2003 Win=65536 Len=0
  9 0.80236100 192.168.1.20        192.168.1.10   TCP     54 c3 > http [FIN, ACK] Seq=387 Ack=2003 Win=65536 Len=0
```

Figure 16: Client – Server Communication Interception

Upon following this TCP stream, valuable information related to this packet was obtained as demonstrated by Figure 17.

4.10 Plaintext Client-Server Communication

This figure shows the TCP packet in pain text. Note that Figure 17 essentially displays the contents of the table in Figure 12. To do so, it uses the HTML tags for table creation *(<table>)*, table rows *(<tr>)*, table data *(<td>)*.



```
<!--
.style5 {font-size: 18px;
.color: #0000FF;
.font-weight: bold;
}
.style1 {.color: #00FF00;
.font-weight: bold;
.font-size: 36px;
}
-->
</style>
</head>

<body>
<p align="center"><span class="style1">CLIENT-SERVER CHAT ROOM POST RETRIEVE... </
span></p>
<hr />
<p><span class="style5">
  <center><table width='900' border='5' cellspacing='3' cellpadding='6'><th> USERNAME</
th> <th> CHAT CONTENT</th><tr><td>SERVER<td>HI SIR, BLACKOUT!<br /
><tr><td>SERVER<td>SO, SAD BUILDING COLLAPSES KILLING THREE!<br /
><tr><td>SERVER<td>MSC. IT SECURITY & AUDIT IS FUN..<br /><tr><td>SERVER<td>HI SIR<br /
><tr><td>SERVER<td>HI SIR<br /><tr><td>SERVER<td>RTRTR<br /
><tr><td>SERVER<td>RTRTR<br /><tr><td><td><br /><tr><td>SERVER<td>RTRTR<br /
><tr><td><td><br /><tr><td>SERVER<td>DFDF<br /></table></span></p>
<p><a href="http://192.168.1.20/client/Home.php">Back</a></p>
<hr />
<p class="style5">&nbsp;</p>
</body>
</html>
```

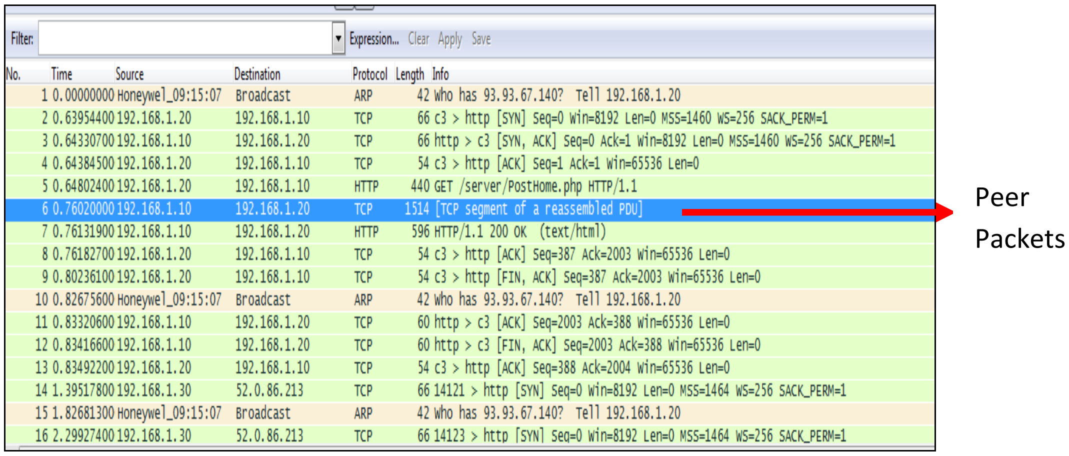
Figure 17: Plaintext Client-Server Communication

Towards the end of this TCP stream, the uniform resource locator (URL) of the machine towards which this request is directed to is given: *http://192.168.1.20/client/home.php* , which was the address of the client machine connecting to the server.

Obviously, this plain text packet reveals enough information that may facilitate further attacks such as spoofing attacks and denial of services (DOS) since the IP addresses and the URLs are evident from this capture. To investigate the server-client communication, the packet in Figure 18 was intercepted and followed.

4.11 Server - Client Communication Interception

To demonstrate the insecure request, the analysis was done from one the peers under domain 192.168.1.20 as shown in figure 18.



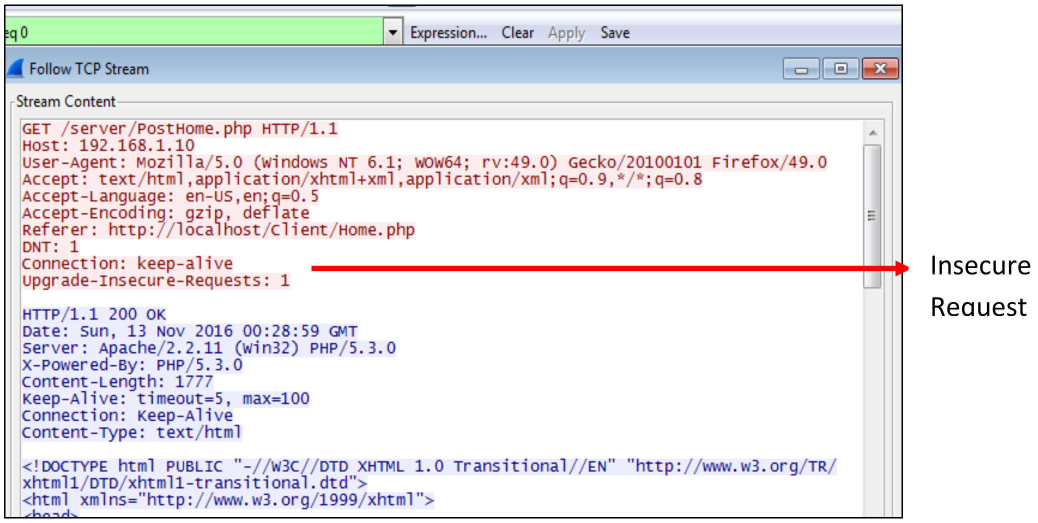
```
No.   Time       Source             Destination   Protocol Length Info
  1 0.00000000 Honeywel_09:15:07 Broadcast     ARP        42 who has 93.93.67.140?  Tell 192.168.1.20
  2 0.63954400 192.168.1.20      192.168.1.10  TCP        66 c3 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
  3 0.64330700 192.168.1.10      192.168.1.20  TCP        66 http > c3 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
  4 0.64384500 192.168.1.20      192.168.1.10  TCP        54 c3 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
  5 0.64802400 192.168.1.20      192.168.1.10  HTTP      440 GET /server/PostHome.php HTTP/1.1
  6 0.76020000 192.168.1.10      192.168.1.20  TCP      1514 [TCP segment of a reassembled PDU]
  7 0.76131900 192.168.1.10      192.168.1.20  HTTP      596 HTTP/1.1 200 OK  (text/html)
  8 0.76182700 192.168.1.20      192.168.1.10  TCP        54 c3 > http [ACK] Seq=387 Ack=2003 Win=65536 Len=0
  9 0.80236100 192.168.1.20      192.168.1.10  TCP        54 c3 > http [FIN, ACK] Seq=387 Ack=2003 Win=65536 Len=0
 10 0.82675600 Honeywel_09:15:07 Broadcast     ARP        42 who has 93.93.67.140?  Tell 192.168.1.20
 11 0.83320600 192.168.1.10      192.168.1.20  TCP        60 http > c3 [ACK] Seq=2003 Ack=388 Win=65536 Len=0
 12 0.83416600 192.168.1.10      192.168.1.20  TCP        60 http > c3 [FIN, ACK] Seq=2003 Ack=388 Win=65536 Len=0
 13 0.83492200 192.168.1.20      192.168.1.10  TCP        54 c3 > http [ACK] Seq=388 Ack=2004 Win=65536 Len=0
 14 1.39517800 192.168.1.30      52.0.86.213   TCP        66 14121 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1464 WS=256 SACK_PERM=1
 15 1.82681300 Honeywel_09:15:07 Broadcast     ARP        42 who has 93.93.67.140?  Tell 192.168.1.20
 16 2.29927400 192.168.1.30      52.0.86.213   TCP        66 14123 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1464 WS=256 SACK_PERM=1
```

Figure 18: Server - Client Communication Interception

Once again, upon TCP stream following of this packet, the information shown in Figure 19 was obtained.

4.12 Plaintext Server-Client Communication

The information obtained showed that the response was using the "GET" form submission method as indicated by the first line. Note that both the client and server request are marked as being *'Insecure-Request'* as demonstrated by line 10, since the request are in plain text.

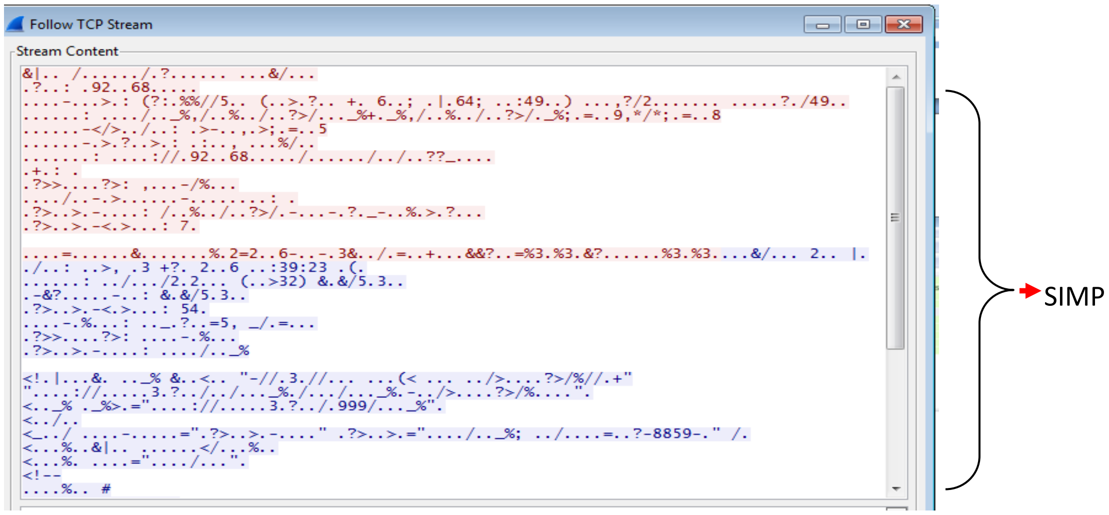


```
GET /server/PostHome.php HTTP/1.1
Host: 192.168.1.10
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/Client/Home.php
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1

HTTP/1.1 200 OK
Date: Sun, 13 Nov 2016 00:28:59 GMT
Server: Apache/2.2.11 (Win32) PHP/5.3.0
X-Powered-By: PHP/5.3.0
Content-Length: 1777
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Figure 19: Plaintext Server-Client Communication

4.13 Scrambled Instant Message Packet

The proposed port-based algorithm would scramble the packets such that the interception of this packet makes no sense to the intruder. Figure 20 shows such kind of TCP packet scrambling in a typical instant message.



```
&|.. /....../.?...... ...&/...
.?..: .92..68.....
....-...>.: (?:.%%//5.. (..>.?.. +. 6..; .|.64; ..:49..) ...,?/2....... .....?./49..
......: ..../.._%,/..%../..?>/..._%+._%,/..%../..?>/._%;.=..9,*/*;.=..8
......-</>../..: .>-..,.>;.=..5
......-.>.?..>.: .:.., ...%/..
.......: ....://.92..68..../....../../..??_....
.+.: .
.?>>....?>: ,...-/%...
..../..-.>......-........: .
.?>..>.-....: /..%../..?>/.-...-.?._-..%.>.?...
.?>..>.-<.>...: 7.

....=......&.......%.2=2..6-..-.3&../.=..+...&&?..=%3.%3.&?......%3.%3....&/... 2.. |.
./..: ..>, .3 +?. 2..6 ..:39:23 .(.
......: ../.../2.2... (..>32) &.&/5.3..
.-&?.....-..: &.&/5.3..
.?>..>.-<.>...: 54.
....-.%...: .._.?..=5, _/.=...
.?>>....?>: ....-.%...
.?>..>.-....: ..../.._%

<!.|...&. .._% &..<.. "-//.3.//... ...(< ... ../>....?>/%//.+"
".....://.....3.?../../..._%./.../..._%.-../>....?>/%....".
<.._% ._%>.=".....://.....3.?../.999/..._%".
<../..
<_../ ....-.....=".?>..>.-...." .?>..>.="..../.._%; ../....=..?-8859-." /.
<...%..&|.. ......</...%..
<...%. ....="..../...".
<!--
....%.. #
```

Figure 20: Typical Scrambled Instant Message Packet

This figure confirms the fact that if instant messages could be scrambled, then t its content is meaningless to the human readers. In such a case, a decryption key will be required to turn this data into human readable format.

V. CONCLUSIONS

This paper sought to demonstrate the fact that instant messaging applications have numerous vulnerabilities, one of them being the transmission of messages in plaintext. To illustrate this, a prototype was developed in Java programming language using its networking sockets. The prototype was run in an Oracle VirtualBox VM environment. The results that were obtained demonstrate clearly that the instant messages are in plaintext in both the sending and receiving machines.
Moreover, data capture that was performed using Wireshak further reveals that the instant messages were not immune from interception and remote monitoring. Therefore, the researchers recommend secure protocols, strong authentication and message encryption at both the receiver and sender so that these messages are in human unreadable format in both terminals. In this way, eavesdropping and remote monitoring of the communication can be thwarted.

REFERENCES

1] Weissbrot & Alison (2016). *Car Service APIs Are Everywhere, But What's In It For Partner Apps?* AdExchanger. ad exchanger.

2] Mark (2015). Private, Partner or Public: Which API Strategy Is Best For Business?. Programmable Web.

3] Z. Vinnie & G. Belvin (2012). *Silent circle instant messaging protocol.*

4] S. Lin Z. Hao; X. Tao; L. Mingshu (2016). *An Empirical Study on Evolution of API Documentation.* International Conference on Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg.

5] O. Wendell (2013*). Cisco CCENT/ CCNA ICND1 100-101 Official Cert Guide*. Pearson Education. pp. Ch. 1

6] M. Green (2014). *Noodling about IM protocols.*

7] Andreas & Buchenscheit (2014). *Privacy implications of presence sharing in mobile messaging applications.* Proceedings of the 13th International Conference on Mobile and Ubiquitous Multimedia. ACM.

8] P. Jagwani (2016). *Analyzing Instant Messaging Applications for Threats : WhatsApp Case Study*. Department of Computer Science, Aryabhatta College, University of Delhi, Delhi (India).

9] T. Frosch C. Mainka, C. Bader, F. Bergsma, J. Schwenk, T. Holz (2016). *How Secure is TextSecure.* Ruhr University Bochum.

10] B. Smith (2013). *UDP vs TCP security*

11] *G.* Fahrnberger (2014). *SIMS: A Comprehensive Approach for a Secure Instant Messaging Sifter.*