International Journal of Science and Engineering Applications
Volume 4 Issue 4, 2015, ISSN-2319-7560 (Online)

A Comparative Study Of Remote Access Technologies and Implementation of a Smartphone App for Remote System Administration Based on a Proposed Secure RFB Protocol

Ernest D. Ganaa
School of Applied Science & Technology
Department of Information & Communication Technology
Wa Polytechnic
Wa, Ghana

Frimpong Twum
College of Science
Dept. of Computer Science
Kwame Nkrumah University of Science & Technology
Kumasi, Ghana

J. B. Hayfron- Acquah
College of Science
Dept. of Computer Science
Kwame Nkrumah University of Science & Technology
Kumasi, Ghana

Abstract: In this paper we proposed advanced technologies to provide tremendous support for network administrators by implementing a secure remote system administration app that runs on android smartphones to aid them administer their servers remotely when they (network administrators) are out stationed using their smartphones.
The android app developed in eclipse establishes a secure connection with a remote server running a PHP application. The app was developed based on the Remote Frame Buffer (RFB) protocol. The RFB protocol, a display protocol has some security lapses including being vulnerable to Man-In-The-Middle (MITM) attack using a few tools and techniques. This research therefore incorporated a self-signed Secure Socket Layer (SSL) certificate in the android app to enable secure encrypted connections to be established between the android app and the remote server to ensure end-to-end security against attacks such as Man-In-The-Middle (MITM).
The whole system was deployed based on client-server architecture with the hand-held smart devices as clients, providing real-time network access to network administrators to their remote servers.
The secure RFB protocol proposed and implemented in the android app was compared with other existing software for remote system administration such as Remote Desktop (RDP), and RFB protocols using ICMP ping command. The results show that the average response time of the RDP protocol was 436ms, that of the RFB protocol was 496ms and that of the android app which is based on a proposed secure RFB protocol was 474ms.

Keywords: Remote access; Remote Frame Buffer; Remote Desktop Protocol; Android app; Remote server; Remote system administration

1. INTRODUCTION

There are several situations where network administrators are faced with the problem of monitoring and administering their various computer networks while away from their offices. In such circumstances, the network administrators depend on third party reports to know the status of their networks. Some even have to direct such third parties as to how to resolve network issues on their behalf when they (network administrators) are out stationed which most times leads to networks jamming and other related issues.

This research looks at remote access technologies and how to implement a smartphone app for remote system administration so as to create some kind of virtual office(s) for System Administrators who will like to always be tied up to their networks even if they are out stationed.

Also, in today's world, it is not a question of whether a remote access software will be used, but which one of the available remote access software will be selected by users and how the selected product will be configured to minimize security risk. It is against this background that this research seeks to conduct a comparative study on remote access technologies in short messaging system (SMS) to perform system administration tasks.

This research work intends to build an android app based on a proposed secure RFB protocol sitting on a smart device that will communicate with a network server running a Hypertext Pre-Processor program. The proposed android app which will act as an interface to the network server will connect to the server using Virtual Private Network (VPN) technology.

The server will perform the processing and send responses back to the android app. As an example, android app will be responsible for issuing basic commands like creating files and as well as performing basic server management task such as creating users, setting user privileges, etc.

2. AIM OF RESEARCH

The aim of this research is to do a comparative study of remote access technologies and implement a smartphone app for remote system administration based on a proposed secure RFB protocol.

The specific objectives are as follows:

- ✓ To investigate what remote access technologies are available
- ✓ To examine the current usage of remote access technologies

- ✓ To evaluate the security of current remote access technologies
- ✓ To implement a smartphone app for remote system administration based on a proposed secure RFB protocol
- ✓ To evaluate the performance of existing remote access technologies as against a proposed secure RFB protocol.

3. SYSTEM OVERVIEW

The purpose of the system is to allow system administrators monitor and administer their computer networks remotely using their android smartphones.

With this system, a system administrator can create a user remotely, create, view and modify text files remotely, check network status, shutdown a server and set user privileges.

The system was developed based on a proposed secure RFB protocol with self-signed Secure Socket Layer (SSL) certificate incorporated into this RFB protocol to ensure end-to-end encrypted connections between the smart device (client) and server.

The system can be used to monitor and administer only one server at a time.

3.1 System Architecture

The system is client-server architecture with the smartphone as the client and the remote computer to be administered as the server.

The android app runs on a smartphone and it is responsible for issuing commands to the remote server. The android app acts as an interface to the remote server.

The restful PHP application runs on the remote server machine and is responsible for processing and returning all requests to the smart device. The database keeps track of all administrators who logon to the system and the activity performed for audit trail purposes.

Figure 1 shows the block diagram of the system, figure 2 shows the system architecture and figure 3 shows the flow of the system.

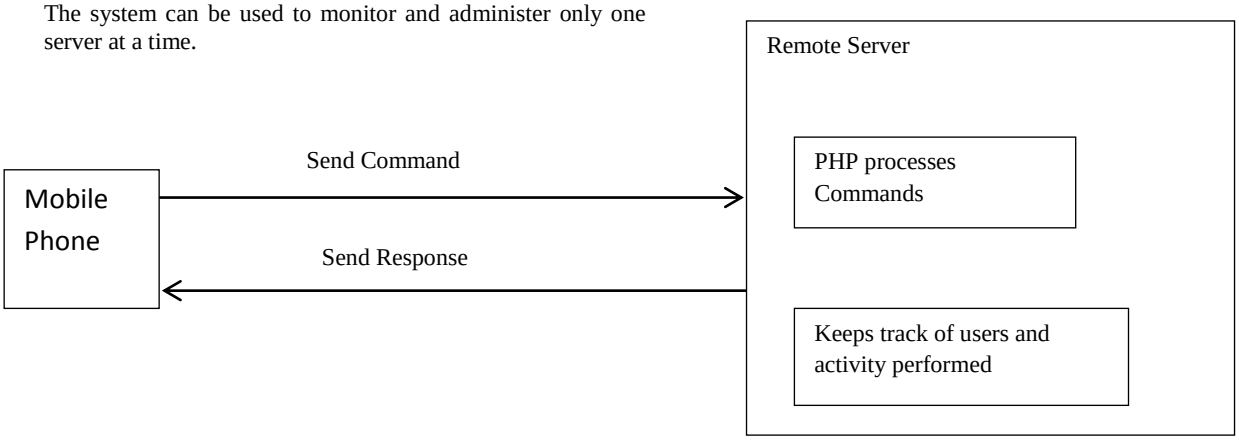

Figure 1. Block Diagram of System. Source Authors' Construct 2015

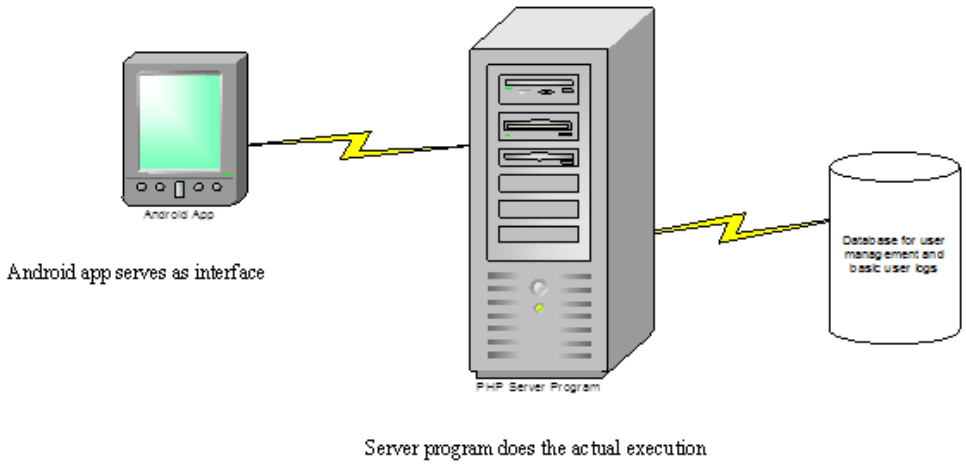

Figure 2. Architecture of system. Source Authors' construct 2015

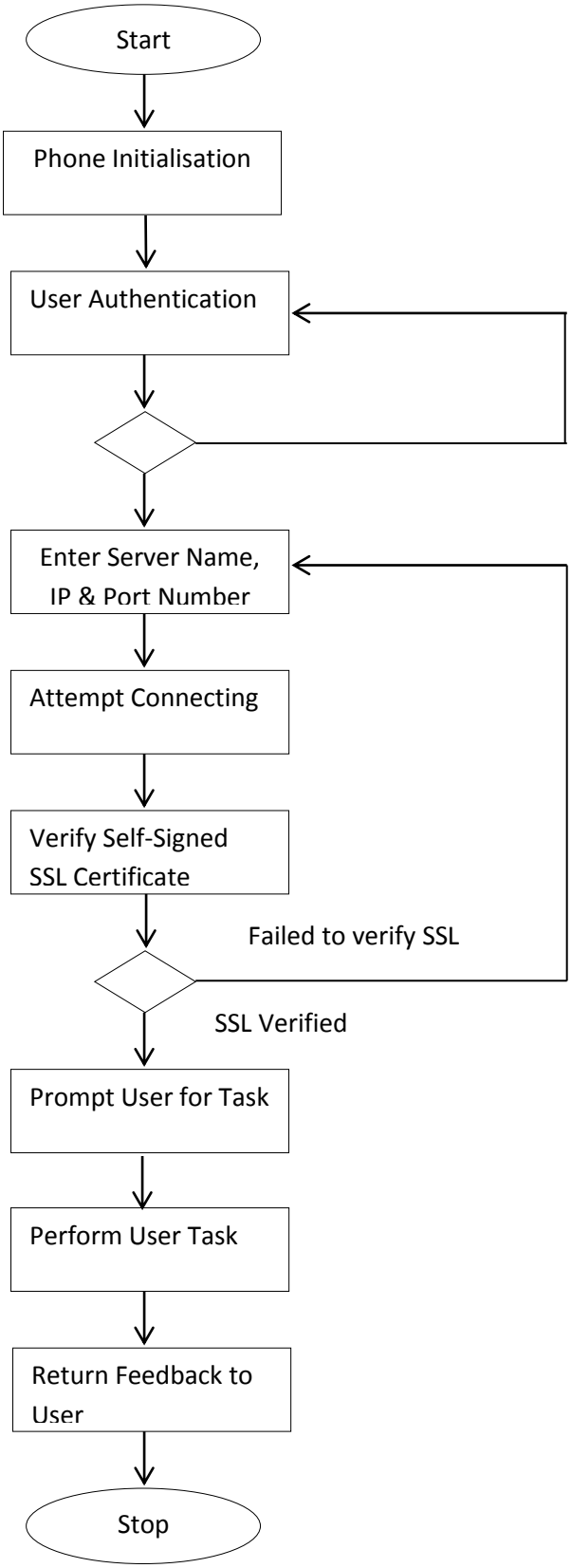

Figure 3: Flow of system. Source: Authors' construct 2015.

3.2 Evaluation of existing Remote Access Technologies

The All remote access systems or applications are developed based on existing and or appropriate technology or technologies. Some existing technologies available for developing remote access systems are:

i. The Remote Frame Buffer (RFB) protocol

The RFB protocol is a simple protocol for sending graphics to be displayed on a remote display or screen. RFB protocol places very little demand on the remote display in terms of processing power and memory demands since all processing is done at the server side. This protocol is a true thin client protocol because it has very low bandwidth requirements and shifts all processing demands to the RFB server instead of the RFB client (Kerai, 2010). The major interest in designing this protocol is to make very few requirements of the client in terms of processing (Richardson, 2010).

The two remote endpoints in the RFB protocol are referred to as the RFB client or viewer and the RFB server (Baig et al., 2012). It works by simply taking rectangles of screen data from the RFB server with a given position and size and puts them into its frame buffer so that they appear in the correct place on the RFB client's screen (Masthan et al., 2013).

Despite the fact that RFB protocol uses encrypted passwords and network, any communication over the network is vulnerable and can be attacked by a Man-In-The-Middle (MITM) by using a few tools and techniques (Kerai, 2010). Also, the applications of VNC which are developed based on RFB protocol are generally slower, offer fewer features and security options than Remote Desktop (RD) which is based on the RDP protocol (Masthan et al., 2013).

Virtual Network Computing (VNC) was developed based on the RFB protocol (Baig et al., 2012).

ii. The Remote Desktop Protocol (RDP)

RDP is a proprietary protocol designed by Microsoft for remote input and display of host running the windows operating systems which is based on the Multipoint Application Sharing (T.128) recommendation by Telecommunication Union (Youming, 2013).

By default, the data that travels between the terminal server and the client is protected by the RC4 symmetric encryption algorithm which provides three levels (high level, medium level and low level) of security (Kerai, 2010). The high level security encrypts data sent from the client to the server using a 128 bit key and does same to data sent from server to client, the medium level security encrypts both data sent from client and server using a 56 bit key if the client is using at least windows 2000 and low level security only encrypts data sent from client to server using 56 bit key or 40 bit key.

According to Montoro (2005), though the data sent between the server and client is encrypted, the RDP protocol may be prone to Man-In-The-Middle attack because there is no verification of the identity of the server when setting up the encryption keys for a session.

The MITM attack works as follows:

- ✓ When the client connects to the server, by DNS spoofing (making a DNS entry point to another IP

address than it was supposed to point to) or ARP poisoning (entering a fake IP address in a host ARP table) which causes diversion of traffic to a different host, the client is fooled to connect to the MITM instead and the MITM in turn sends a request to the server. This involves maliciously modifying the relation between an IP address and its matching MAC address (Behboodian & Razak, 2011)

- ✓ Through this, the server then sends its public key, in clear text through the MITM and the MITM now sends the packet further to the client, but exchanging the public key with another one for which it knows the private part.
- ✓ The client upon receiving this sends a random salt, encrypted with the server's public key, to the MITM. The MITM decrypts the clients random salt with its private key, encrypts it with the real servers public key and sends it back to the server.
- ✓ The MITM now knows both the server and the client salt, which is enough information to construct the session keys used for further packets sent between the client and the server.

This vulnerability occurs because the client by no means will try to verify the public key of the server. In other protocols such as the Secure Shell protocol (SSH), most client implementations solve this MITM attacks by allowing the user to answer a question whether a specific server's key fingerprint is valid (Montoro, 2005).

Microsoft confirmed the above problem and fixed the new versions of Remote Desktop Clients. Recent clients now check the Terminal Server's identity to verify its public key before allowing connections. The implication of this is that Remote Desktop has a very strong security, but of course, as time passes by, attackers develop more sophisticated tools to break through. Nam et al. (2012) also stated that ARP (Address Resolution Protocol) poisoning can be resolved by delivering the public key and the MAC address of the server to the client, however this is to be set by the network administrator manually.

Remote Desktop Protocol (RDP) 6.0 supports Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols which encrypt data sent between a server and a client (Boling, 2007). Quite a number of the Windows operating systems such as Windows Server 2003 SP1, Windows Server 2008, Windows XP, Windows Vista, Windows 7 and Windows 8 support SSL/TLS for RDP 6.0.

Despite the MITM security problem, RDP is designed to support different types of network topologies, multiple LAN protocols and just like VNC, RDP works on TCP/IP connections (Kerai, 2010).

4. The Proposed Secure RFB Protocol and Implementation of an Andriod Smartphone App for Remote System Administration

Based on the fact that the RFB protocol, a display protocol has some security lapses including being vulnerable to Man-In-The-Middle (MITM) attack using a few tools and techniques, the study in trying to secure the RFB protocol implemented a self-signed Secure Socket Layer (SSL) certificate on top of the RFB protocol. The purpose of this is to enable secure encrypted connections to be established between the android app and the remote server to ensure end-to-end security against attacks such as Man-In-The-Middle (MITM)

Secure Socket Layer (SSL) certificate is a cryptographic protocol that creates an encrypted communication channel between a server and client that makes internet traffic indecipherable to third parties that might intercept them (Roosa & Schultze, 2010).

A self-signed certificate is one that is signed by the individual who created it rather than a trusted Certificate Authority (Code Project, 2014). This research used a self-signed certificate because they are very convenient in mobile development since mobile apps in most cases interact with only one server unlike web browsers. Also, if well implemented, they work like certificates from a certificate authority.

This is how the self-signed certificate was created on the server the android app will be communicating with:

- ✓ A KeyStore is created using "bcprov-jdk15on-146.jar" which is a java class. This class can be downloaded from www.bouncycastle.org/download/bcprov-jdk15on-146.jar. This file will be stored in "C:\androidproject". This file is used to generate the KeyStore.
- ✓ A keytool is then used to generate the key "keytool –genkey -alias androidproject –keystore C:\androidproject/androidprojectsssl.keystore –validity 365.
- ✓ The above generated key is then exported from the .KeyStore file to .cer file using the command "-export –alias androidprojects –keystore C:\androidprojects\androidprojectssl.keystore –file C:\androidproject\androidprojectsslcert –cer"
- ✓ The keystore file is then saved in "/androidappdir/raw/"
- ✓ A class called MyAndroidClient is then written to hardcode the self-signed generated certificate in the android app.

In implementing this self-signed certificate created above in the android app, the study sets up a custom TrustManager that will trust the above self-signed certificate. This custom TrustManager is then provided with custom SSLContext as demonstrated in the code snippet shown in figure 4 below. The idea here is to do certificate pinning with this self-signed SSL certificate by hard-coding the certificate known to be used by the server in the android app so that the app will then ignore the smartphone's trust store and rely on its own trust store thereby terminating connections that do not match with this self-signed certificate.

```
import java.io.InputStream;
import java.security.KeyStore;

import android.content.Context;

public class MyAndroidClient extends DefaultHttpClient {

 private static Context context;

 public static void setContext(Context context) {
  MyAndroidClient.context = context;
 }

 public MyAndroidClient(HttpParams params) {
  super(params);
 }

 public MyAndroidClient(ClientConnectionManager httpConnectionManager, HttpParams params) {
  super(httpConnectionManager, params);
 }

 @Override
    protected ClientConnectionManager createClientConnectionManager() {
        SchemeRegistry registry = new SchemeRegistry();
        registry.register(new Scheme("http", PlainSocketFactory.getSocketFactory(), 80));
        registry.register(new Scheme("https", newSSLSocketFactory(), 443));
        return new SingleClientConnManager(getParams(), registry);
    }

    private SSLSocketFactory newSslSocketFactory() {
        try {
         InputStream in =
MyAndroidClient.context.getResources().openRawResource(R.raw.androidprojectssl);
            try {
              trusted.load(in, "g@n@@1984".toCharArray());
            } finally {
               in.close();
            }
            SSLSocketFactory sf = new SSLSocketFactory(trusted);
            sf.setHostnameVerifier(SSLSocketFactory.STRICT_HOSTNAME_VERIFIER);
            return sf;
        } catch (Exception e) {
            throw new AssertionError(e);
        }
    }
}
```

Figure 4. Code snippet for implementing self-signed SSL certificate

5. Evaluating the performance of the RDP, RFB, and the Proposed Secure RFB Protocol

The performance of the proposed secure RFB protocol implemented in the android app was compared with existing remote access software which are based on the RDP, and the RFB protocols respectively using ICMP ping command. Figure 5 below shows the ICMP ping results for the RDP protocol.

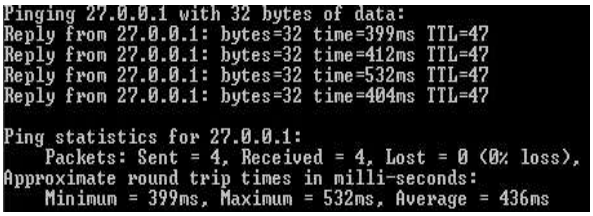

Figure 5: RDP protocol ICMP ping results

Figure 6 below shows the ICMP ping results for the RFB protocol.

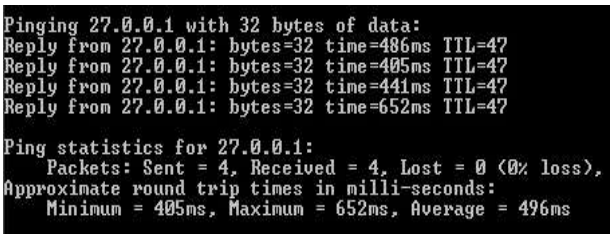

Figure 6: RFB protocol ICMP ping results

Figure 7 shows the ICMP ping results for the secure RFB protocol

Figure 7: The Proposed Secure RFB protocol android app ICMP ping results.

6. Conclusions, Recommendation, and Acknowledgment

From the study, it was revealed through literature that by default the data that travels through the terminal server and client in the case of the RDP protocol is protected by the RC4 symmetric encryption algorithm which provides 3 levels of security. It was further revealed through literature that applications based on RFB protocol offer fewer features and security options than remote desktop which is based on the RDP protocol.

The study has successfully implemented a smartphone app for remote system administration based on a proposed secure RFB protocol. Self-signed SSL certificate was incorporated into the RFB protocol to make it a secure RFB protocol to ensure that secure encrypted connections are established between the smartphone and the server.

This research also revealed that the average response time of the RDP protocol (Remote Desktop) was 436ms, the average response time of the RFB protocol (VNC) was 496ms and the average response time of the android app which is based on a secure RFB protocol was 474ms.

It is recommended that in the future the android app be made to be able to monitor and administer at least two remote servers at a time.

We thank God for his mercies given to us to complete this work.

7. REFERENCES

[1] Baig S, Rajasekar M. & Balaji P. (2012) Virtual Network Computing Based Remote Desktop Access. *International Journal of Computer Science and Telecommunications.* Volume 3, Issue 5. pp. 127.

[2] Behboodian N. & Razak S. A. (2011) ARP Poisoning Attack Detection and Protection in WLAN via Client Web Browser. *International Conference on Emerging Trends in Computer and Image Processing.* pp. 20.

[3] Boling D. (2007) Windows Embedded CE 6.0 R2 Remote Desktop Protocols and Internet Explorer. pp. 6. Retrieved from: http://download.microsoft.com/download/5/8/e/58e0c008-fc15-4a3b-9728-c0103bab6473/Windows%20Embedded%20CE%206.0%20R2%20Remote%20Desktop%20Protocol%20and%20Internet%20Explorer_whitepaper.pdf.

[4] Code Project (2014) Android Security-Implementation of Self-Signed SSL Certificate for your App. Retrieved from: www.codeproject.com/articles/826045/android-security-implementation-of-self-signed-SSL

[5] Kerai P. (2010) Tracing VNC and RDP Protocol Artefacts on Windows Mobile and Windows Smartphone for Forensic Purpose. *In Proceedings of International Cyber Resilience Conference.* Australia. pp. 58. Retrieved from: http://ro.ecu.edu.au/icr/7

[6] Masthan K, Kumar S. K. & Prasad H. V. (2013) Virtual Network Computing of User Appliances. International Journal of Computer Science and Mobile Computing. Volume 2, Issue 8. pp. 132.

[7] Montoro M. (2005) Remote Desktop Protocol, the Good the Bad and the Ugly. pp. 1-2. Available from: www.oxid.it.

[8] Nam Y. S, Jurayev S, Kim S, Choi K. & Choi S. G (2012) Mitigating ARP poisoning-base man-in-the-middle attacks in wired or wireless LAN. *EURASIP Journal on Wireless Communication and Networking*. pp. 2.

[9] Richardson T. (2010) *The RFB Protocol.* pp. 3. Retrieved from: www.realvnc.com.

[10] Roosa S. B. & Schultze S. (2010) The "Certificate Authority" Trust Model for SSL: A Defective Foundation for Encrypted Web Traffic and a Legal Quagmire. Intellectual Property & Technology Law Journal. Volume 22, Number 11. pp. 3.

[11] Youming L. (2013) *Virtual Networking for Mobile Cloud Computing*. Master's Thesis, Aalto University-Finland. pp. 19. Retrieved from: https://into.aalto.fi/download/attachment/.../Lin_Youming_thesis.pdf?