GENERAL INSTRUCTIONS

This homework has two parts: one part with a set of theory questions and a programming part. Programming can be done in either C or Java. Submissions should include the annotated source code. Programs that do not compile will get a low grade. Make sure your programs do not crash when given bad input, but rather produce a useful warning or error message and take the appropriate action (recover or quit).

SUBMISSION INSTRUCTIONS

The compressed files (either tar or zip) will be submitted using email to astavrou@gmu.edu.

A compressed (zip or tar, and gzipped) file named {your last name}-hw3.tgz or {your last name}-hw3.zip containing a subdirectory for each problem, named p1, p2, etc. Each subdirectory should contain:

• All source code, including any test programs, for the problem, if required.

• A file answering any questions posed.

• Any additional files required.

Please make sure that you send me a single compressed file with all the documents and code. The compressed file should not exceed 1MB.
THEORY/WRITTEN QUESTIONS (80 points)

1) List all the different types of Denial of Service (DoS) attacks for networked applications and mechanisms to defend against them.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth / DDoS</td>
<td>Bandwidth shaping&lt;br&gt;Over-provisioning&lt;br&gt;Network Capabilities&lt;br&gt;Anomaly Filtering&lt;br&gt;Black-Hole Routing&lt;br&gt;Pushback</td>
</tr>
<tr>
<td>CPU</td>
<td>Rate-limiting requests&lt;br&gt;Challenge-response before doing work&lt;br&gt;Computational Puzzles&lt;br&gt;Network Capabilities</td>
</tr>
<tr>
<td>Memory</td>
<td>Anti-spoofing – Blacklisting site origins*&lt;br&gt;Improved data structures – compact packets&lt;br&gt;SYN cookies&lt;br&gt;Higher limit of half-open connections&lt;br&gt;Just-in-Time (JIT) state creation</td>
</tr>
<tr>
<td>General:</td>
<td>Coding practices and testing&lt;br&gt;Hardening services</td>
</tr>
<tr>
<td>Bayesian Filter</td>
<td></td>
</tr>
<tr>
<td>Anti-spoofing</td>
<td></td>
</tr>
<tr>
<td>Program Availability</td>
<td>Exploiting bugs to crash it</td>
</tr>
<tr>
<td>Availability</td>
<td></td>
</tr>
<tr>
<td>Relector</td>
<td>Small impetus causes multiplied effect</td>
</tr>
</tbody>
</table>

2) Why do we implement several different forms of encryption when we protect Voice over IP (VoIP) systems and more specifically SIP?

There are 3 different segments of a VoIP call: the setup (control signals); the call, itself (the data); the billing (accounting information). The different phases require different forms of security given their disparate natures, the source-destination of the call (e.g. intranet vs. internet), and due to the fact there is no real PKI for SIP and VoIP. IPSec may be used to encrypt all data, but is problematic, unless all the traffic exists on a homogeneous network structure, e.g. between organization and SIP gateway. Alice and Bob need to validate each other before they begin
3) How do we protect wired communications against eavesdropping?

Encryption does not prevent a user from eavesdropping, but from deciphering the data she has gathered. If the attacker sits on the same wire as the attackee, she will receive all the packets on that wire, including her target. However, if they’re segmented, even within the same subnet\LAN, using selective hardware, sniffing may be mitigated.

If, in an Ethernet, star topology, the attacker and target are on the same (non-intelligent) hub, eavesdropping is unpreventable. If, however, a switched network is used, the attacker may receive only the packets destined for her, when packets are associated by MAC, i.e. the packets are passed through the ports associated with a particular host. N.B. Attacks exist to overload a switch and render it no better than a hug. In addition MACs can be spoofed, but no same MAC address should occur on two different ports, at the same time- may be indicative of attack.

4) What are the current security problems with Domain Name Service? Are there any proposed solutions?

There are several exploits on DNS:
• DNS root attack – (D)DoS on root and TLD servers caused minimum delays due to long (Time-to-Live) TTLs and DNS caching; the significant, .org, TLD server delay was unaddressed
• DNS (DDoS) Amplification attacks – when an attacker issues a small DNS query with the intent of provoking a larger, recursive query (great traffic) to a root or TLD server; possible suggestions to mitigate this are (1) source address verification, (2) blocking and filtering, (3) secure configuration of DNS servers, and (4) disabling open, recursive DNS
• DNS Reflector Attacks – reminiscent of previous attack, using a compromised DNS and a BIG TXT record, and other DNS servers that employ open recursion, but it differs in that the victim is a user whose IP has been spoofed, and DNS queries sent out to the servers, causing a DoS when s/he receives the response; suggested solutions for the victim are to filter; however,
the DNS solutions are relatively the same as above, but with added suggestions, like specialized hardware (SOHO using proxy code)

Also consider:
• DNS poisoning – tricking a DNS server into caching an incorrect record, by exploiting protocols lacking validation processes
• Remote stack overflow
• Fast Flux – changing the A (and NS) records for a static address (or namespace) after short time periods (like 30 minutes), such that they point to different hosts

5. How do we protect wireless communications against eavesdropping? Is WEP a good solution?

On the wireless router:
• Turning off the SSID broadcast
• Enabling MAC filtering
• Requiring a network key
• Using encryption: WEP (bad), TKIP (WPA, WPA2), etc.

WEP is cryptographically insecure. In fact, there are widely available programs that can crack WEP in as little as a couple minutes, with well-powered setups, or in short of 10mins, with systems typical of today’s desktop. E.g. WEPCrack or Aircrack (available in suites like Backtrack), using a tool like Airsnort to capture packets.

The insecurity steams from: (1) the use of RC4, which is a stream cipher, on a datagram-oriented standard; (2) as an extension of (1), all users share the same key, and rekeying is infeasible; (3) a limited IV space of 24 bits, which leaves it vulnerable to cryptanalysis and replay, and; (4) use of CRC to checksum, which might allow an attacker to acceptably modify a packet (to redirect or inject false information).

**PROGRAMMING (20 points)**

John Supercoder generated a very simple SSL server:

http://cs.gmu.edu/~astavrou/courses/isa_656_S08/isa656server.java

http://cs.gmu.edu/~astavrou/courses/isa_656_S08/isa656client.java
Unfortunately, his code has some errors both in terms of syntax and in terms of logic. Your task is to help John debug and test his code (prove that it is running).

(Hint: start with the syntax errors and then locate the logical ones).

**CLIENT**

```java
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.io.*;

public class isa656client {
    public static void main(String[] args) {
        //System.out.println("Client is running");
        try {
            SSLSocketFactory sslsocketfactory = (SSLSocketFactory)
                    SSLSocketFactory.getDefault();
            SSLSocket sslsocket = (SSLSocket)
                    sslsocketfactory.createSocket("localhost", 9999);
            final String[] suites = {
                "SSL_DH_anon_WITH_RC4_128_MD5"};
            sslsocket.setEnabledCipherSuites(suites);

            InputStream inputstream = System.in;
            InputStreamReader inputstreamreader = new
                    InputStreamReader(inputstream);
            BufferedReader bufferedreader = new
                    BufferedReader(inputstreamreader);

            OutputStream outputstream = sslsocket.getOutputStream();
            OutputStreamWriter outputstreamwriter = new
                    OutputStreamWriter(outputstream);
            BufferedWriter bufferedwriter = new
                    BufferedWriter(outputstreamwriter);

            String string = null;
            while ((string = bufferedreader.readLine()) != null) {
                string += 

            }
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}
```
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import java.net.*;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;

public class isa656server {
    public static void main(String[] args) {
        try {
            SSLServerSocketFactory sslserversocketfactory = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
            SSLServerSocket sslserversocket = (SSLServerSocket) sslserversocketfactory.createServerSocket(9999);
            final String[] suites = {
                "SSL_DH_anon_WITH_RC4_128_MD5"
            };
            sslserversocket.setEnabledCipherSuites(suites);
            SSLSocket ssllsocket = (SSLSocket) sslserversocket.accept();

            InputStream inputstream = ssllsocket.getInputStream();
            InputStreamReader inputreader = new InputStreamReader(inputstream);
            BufferedReader bufferedreader = new BufferedReader(inputreader);
            String string = null;
            while ((string = bufferedreader.readLine()) != null) {
                System.out.println(string);
                System.out.flush();
            }
        } catch (SocketException se) {
            System.out.println("Client disconnected.");
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}