2016

GENERIC ALGORITHM BASED INTRUSION DETECTION SYSTEM (GA-IDS)

Documentation

This documents the working process of the GA-IDS

Oluwole Oyetoke
oluwoleoyetoke@gmail.com
3/2/2016

Table of Contents

INTRODUCTION

The GA-IDS is a full-fledged host based intrusion detection system developed using the Java programming language to help detect packets having spoofed IP addresses. It first and foremost sniffs the incoming packets on the host system and there after analyzes them in order to detect an intrusion. Considering the fact that this sniffing process is a low level operation, the java application makes use of the Java Packet Capturing Library (JpCap) which works in conjunction with the Windows Packet Capturing Library (WinpCap).

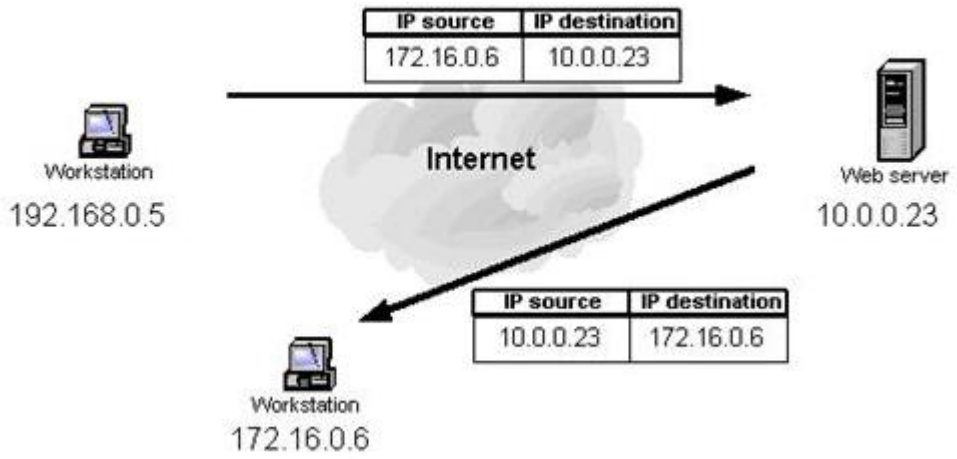

Figure 1: Diagram showing IP Spoofing

ABOUT JPCAP

JpCap is an open source network packet capture library based on the LibpCap and WinpCap libraries. It is usable with Java to capture and display network traffic on LINUX, Windows and Macintosh computers. JpCap captures the following types of packets and can even analyze each packets header and data payload.

- Ethernet
- TCP
- UDP
- IPv4
- IPv6
- ARP/RARP
- ICMPv4 packets

JpCap captures raw packets live from the wire, automatically identify its packet types and generate corresponding Java objects. It can also filter the packets according to

user's specified rules before dispatching them to the application. JpCap can also send raw packets to the network, save and read captured packets to and from an offline file.

HOW THE GA-IDS WORKS

The GA-IDS works in a 5-step cycle. The following explicate the GA typical operation loop.

1. Load and Select Available Network Interfaces on the Computer

As we know, to capture packets from a network, the first thing one has to do is to obtain the list of functioning network interfaces on the computer. To do so, JpCap provides JpcapCaptor.getDeviceList() method. It returns an array of Network interfaces objects. Therefore, the first important operation the system performs is to allow the user load the available network interfaces on the computer so that he/she can choose the desired interface whose packets are to be sniffed and analyzed. The java class method written in codebase 1 below helps carry out this operation.

Codebase 1: *Java Method to Open Available Interfaces*

```
public synchronized void GetAvailableInterfaces() {

//Obtain List of Network Interfaces/JpcapInstance
      IDS.Interfaces = JpcapCaptor.getDeviceList();
      IDS.TotalNumberOfInterfaces =
IDS.Interfaces.length;
      IDS.MyMacAddresses = new
byte[IDS.TotalNumberOfInterfaces][];
      Vector list = new Vector();
      list.clear();
      MainFrame.TextArea.setText("");
      int counter = 0;
      for (int i = 0; i < IDS.Interfaces.length; i++) {
         counter = counter + 1;

         print.TextAreaAppend("------------------------
INFORMATION ON NETWORK INTERFACE " +
counter + "------------------------");

         print.TextAreaAppend("\nName: " +
IDS.Interfaces[i].name);
         print.TextAreaAppend("\nDataLink Name: " +
IDS.Interfaces[i].datalink_name);

         print.TextAreaAppend("\nDataLink
Description: " +
IDS.Interfaces[i].datalink_description);
         print.TextAreaAppend("\nGeneral
Description: " + IDS.Interfaces[i].description);
         list.add("INTERFACE " + counter + ": " +
IDS.Interfaces[i].description);
         print.TextAreaAppend("\nLoop Back: " +
IDS.Interfaces[i].loopback);
         print.TextAreaAppend("\nIP Address: ");
         for (NetworkInterfaceAddress c :
IDS.Interfaces[i].addresses) {

            print.TextAreaAppend(c.address.toString());

         }

          // NetworkInterfaceAddress[] b =
IDS.Interfaces[i].addresses;
         //
print.TextAreaAppend(b[0].address.toString());
         print.TextAreaAppend("\nMAC Address: ");
         for (byte c : IDS.Interfaces[i].mac_address) {

print.TextAreaAppend(Integer.toHexString(c & 0xff)
+ ":");
         }
      // IDS.MyMacAddresses[i][0] =
IDS.Interfaces[i].mac_address;
         print.TextAreaAppend("\n");
         print.TextAreaAppend("\n");
      }

      MainFrame.InterfacesList.setListData(list);

   }
```

End of Code

After the interface opening process, the user is then allowed to select the desired interface to sniff or the combination of interfaces to sniff. The selection processes simples makes the system obtain an instance of the JpcapCaptor as can be seen in line 1 of codebase 1.

2. Begin Sniffing Operation

At activation i.e once you obtain an instance of JpcapCaptor, you can capture packets from the interface. There are two major approaches to capture packets while using a JpcapCaptor instance and they are

a. Using a call-back method
b. Capturing packets one by one.

Using a Call-back method

In this approach, you implement a call-back method to process captured packets, and then pass the call-back method to JpCap so that JpCap calls it back every time it captures a packet. Let's see how you can take this approach in detail.

First, you implement a call-back method by defining a new class which implements the PacketReceiver interface. The PacketReceiver interface defines a receivePacket() method, so you need to implement a receivePacket() method in your class. The following class implement a receivePacket() method which simply prints out a captured packet.

Codebase 2: java Class which implementsthe PacketReceiver Interface

```
class PacketPrinter implements PacketReceiver {

//this method is called every time Jpcap captures a packet
public void receivePacket(Packet packet) {

//just print out a captured packet
System.out.println(packet);

}

}
```

End of Code

Once the class in the codebase 2 above has been set up, then, you can call either JpcapCaptor.processPacket() or JpcapCaptor.loopPacket() methods to start capturing using the callback method. When calling processPacket()or loopPacket() method, you can also specify the number of packets to capture before the method returns. You can specify -1 to continue capturing packets infinitely.

Codebase 3: Code to Capture/Sniff Traffic

```
public void Capture(){

JpcapCaptor captor=JpcapCaptor.openDevice(device[index], 65535, true, 5000);

while (true){
//captures 10 packets before ending
captor.processPacket(10,new PacketPrinter());
//To capture packets unending, change value '10' above to '-1'
captor.close();
}


}
```

End of Code

The two methods for callback, processPacket() and loopPacket(), are very similar. Usually you might want to use processPacket() because it supports timeout and non-blocking mode, while loopPacket() doesn't.

Capturing Packets One by One

Capturing packets one by one Using a callback method is a little bit tricky because you don't know when the callback method is called by Jpcap. If you don't want to use a callback method, you can also capture packets using the JpcapCaptor.getPacket() method. getPacket() method simply returns a captured packet. You can (or have to) call getPacket() method multiple times to capture consecutive packets

Codebase 4: Java method showing how to capture traffic one-by-one

```
public void CaptureOneByOne{

JpcapCaptor captor=JpcapCaptor.openDevice(device[index], 65535, true, 5000);

for(int i=0;i<10;i++){
//capture a single packet and print it out
System.out.println(captor.getPacket());
}
captor.close();

}
```

End of Code

3. Initialize IP Spoof Detection

The GA-IDS performs the spoof detection using the generic algorithm. The generic algorithm simply makes use of chromosome scrutinization to detect phenotype changes and mutation. It is believed that every living organism has a blueprint encoded in its gene. The genes are connected together to form a chromosome and these chromosome

come together to make the organism itself otherwise known as the phenotype. In this case, the GA-IDS treats the received packet as a phenotye. It extracts selected genes of the IP packets and then combines these genes into a verification chromosome which is termed the packets's chromosome needed. As long as the IP spoof detection mechanism, the GA-IDS scrutinizes each packet's header and extracts the following attributes/genes

- Source IP
- Source Mac Address
- Initial Time to Live
- Hop Count Protocol Type
- Packet ID

All these genes represent variables/genes that can be tampered with if an intruder is trying to spoof. Therefore, what the IDS does is to convert all of these genes into their binary equivalent and concatenate it to form the chromosome. See chromosome structure below.

Chromosome Structure:

Source MAC Address In Binary

+

Destination MAC Address In Binary

+

TTL In Binary

+

Hop Count In Binary

+

Packet ID Validity;

Chromosome Example

Take for example, a packet with the following details

Source MAC: 02:00:4c:4f:4f:50

Destination Mac: 01:00:5e:7f:ff:fa

TTL: 32

Hop Count: 1

Packet ID: 288

This packet will have the following chromosome:

1001001100100111110011111010000101011110111111111111111111110101111111

Note that the last bit of the chromosome represents Packet ID validity. If the packet's ID is greater than the one previously sent, the packet validity is set to 1 otherwise, it is a. Packet ID is an increasing value, as such, the only reason why a subsequent packet might have a lesser packet ID is mostly because a spoofer is tampering with stuffs from the other end

If the GA-IDS receives a packet for the first time from a particular IP address, it saves its chromosome with 100% fitness. However, the chromosome of subsequent packets receives from this source IP address are compared with the chromosome of the fittest packet in the database. If the fitness level of the new packet received is less than the packet previously received, it indicated a tampering. At this juncture, the GA-IDS alerts the network administrator of an intrusion.

The default minimum allowable fitness level set by the GA-IDS system is 65% chromosome fitness. However, the user can adjust this to suit the network environment.

The diagram below shows you a picture shows you the structure of a typical IP packet header looks like

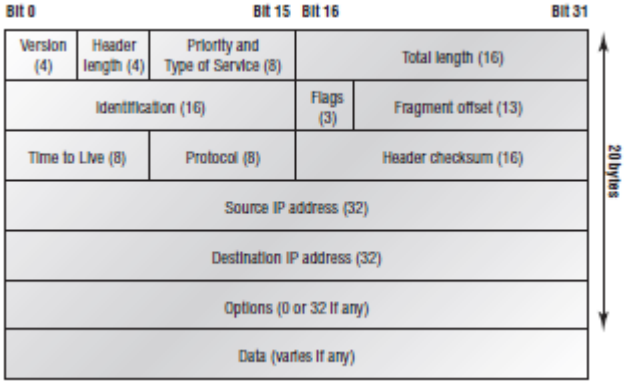

Figure 2: IP Packet Header

Please note that GA-IDS analyzes other packets including ICMP, IGMP, TCP, UDP etc

How Chromosome Fitness Levels Are Generated:

For example, if my system receives a packet from a source IP address for the first time, the GA-IDS extracts the genes as stated above and generates the chromosome. Take for example that a packet's chromosome is: 1010101010. Furthermore, if the system receives another packet from this same source IP address, it is expected that the new chromosome that will be generated be same or nearly same with the previous one because the source MAC address, destination MAC address, inital Time To Live and Hop count be of about the same value. And also, the Packet ID is expected to be greater than

that of the packet previously received. If all of these are true, then the second packet received should have about the same chromosome. However, in the case in which the chromosome of the second packet received is 1100101001, we can see by comparison that

	1	0	1	0	1	0	1	0	1	0
	1	1	0	0	1	0	1	0	0	1
Result:	**M**	**N**	**N**	**M**	**M**	**M**	**M**	**M**	**N**	**N**

Where M = Match

N = No Match

From the above, we can see that there are 4 differences in chromosome, which will mean packet B has been grossly tampered with, having only 60% fitness. At this point, the GA-IDS emailing system will send the administrator an alert message

4. Save to Database

At this juncture, the GA-IDS saves the packet's details into the database so that future incoming packets can be compared with the already saved packets.

SERIAL_ID	SOURCE_IP	SOURCE_MAC	DESTINATIO...	PACKET_ID	PROTOCOL	CHROMOSOME	FITNESS	TIME_STAMP
1	169.254.227.196	02:00:4c:4f:4f:50	01:00:5e:7f:ff:fa	288	UDP	1001001100100111110011111010000101O...	100.0	2016-02-10 11...
2	169.254.227.196	02:00:4c:4f:4f:50	01:00:5e:7f:ff:fa	289	UDP	1001001100100111110011111010000101O...	100.0	2016-02-10 11...
3	169.254.227.196	02:00:4c:4f:4f:50	01:00:5e:7f:ff:fa	290	UDP	1001001100100111110011111010000101O...	100.0	2016-02-10 11...

5. Save To File

The system saves captured packets into a binary file so that you can later retrieve them To save captured packets, the system first opens a file by calling JpcapWriter.openDumpFile() method with an instance of JpcapCaptor which was used to capture packets and a String filename. The code below explains this process explicitly

Codebase 5: Code Showing How JpCap Writes to File

```
JpcapCaptor captor = JpcapCaptor.openDevice(IDS.Interfaces[InterfaceToPrint], 65535,
true, 5000);
       JpcapWriter
writer=JpcapWriter.openDumpFile(captor,"DumpFile"+InterfaceToPrint+".txt");
   writer.writePacket(Pack);
writer.close();
```

End of Code