Journal of Xi'an University of Architecture & Technology ISSN No : 1006-7930

DETECTING THE TRAFFIC IN NETWORK BY MONITORING THE HEADER DATA

E. Bagyalakshmi, M.Sc.,M.Phil,B.Ed
Assistant Professor, Department of Computer Science,
Siga College of management and Computer Science, Kappiyampuliyur

E. Rajalakshmi, M.C.A.,M.Phil
H.O.D, Department of Computer Application,
RAAK Arts and Science College,
Perambai

Abstract— In this paper we detect the traffic anomalies by monitoring the header data. Some attacks like denial of service led to develop the techniques for identifying the network traffic. If we have the efficient analysis tool we could prevent the network from the traffic before it could get attacked. We can analyze the network traffic with the help of, correlation of the destination IP address in the egress router. The address correlations are data transformed using the discrete wavelet transform for detecting the traffic anomalies. Results from trace-driven evaluation suggest that proposed approach could provide an effective means of detecting anomalies close to the source. We also present a multidimensional indicator using the correlation of port numbers and the number of flows as a means of detecting anomalies.

I.INTRODUCTION

Network traffic monitoring or network traffic analysis is a security analytical tool used by computer network security administrators to detect issues that can affect functionality, accessibility, and network traffic security. Therefore, it is a network security technique for checking the network traffic of internet-connected devices, the type of data the devices are retrieving, and the bandwidth level each device is consuming.

Furthermore, network security admins and other certified network defenders use a network security program for carrying out network traffic monitoring tasks. The most frequent attacks on network infrastructure, using denial of service (DoS) attacks and worms, have led to an increased need for developing techniques and monitoring network traffic. If efficient analysis tools were available, it could become possible to detect the attacks, anomalies and take action to suppress them before they have had much time to propagate across the network. In this paper, we find the possibilities of traffic-analysis based mechanisms for attack and anomaly detection. This work especially to reduce attacker may hijack the campus machines to stage an attack on a third party. A campus may want to prevent or limit misuse of its machines in staging attacks, and possibly limit the liability from such attacks. In particular, we studied the utility of observing packet header data of outgoing traffic, such as, port numbers, destination addresses and the number of flows, in order to detect attacks/anomalies originating from the campus at the edge of a campus. Detecting anomalies/attacks close to the source allows us to limit the potential damage close to the attacking machines. Traffic monitoring close to the source may enable the network operator quicker identification of potential anomalies and allow better control of administrative domain's resources. Attack propagation could be slowed through early detection. Our approach passively monitors network traffic at regular intervals and analyzes it to find any abnormalities in the aggregated traffic. By observing the traffic and correlating it to previous states of traffic, it may be possible to see whether the current traffic is behaving in a similar (i.e., correlated) manner. The network traffic could look different because of flash crowds, changing access patterns, infrastructure problems such as router failures, and DoS attacks. In the case of bandwidth attacks, the usage of network may be increased and abnormalities may show up in traffic volume. Flash crowds could be observed through sudden increase in traffic volume to a single destination. Sudden increase of traffic on a certain port could signify the onset of an anomaly such as worm propagation. Our approach relies on analyzing packet header data in order to provide indications of Possible abnormalities in the traffic.

Our approach to detecting anomalies envisions two kinds of detection mechanisms, i.e., postmortem and real-time modes. A postmortem analysis may exploit many hours of traffic data as a single data set, employing more rigorous, resource-demanding techniques for analyzing traffic. Such an analysis may be useful for traffic engineering purposes, analysis of resource usage, understanding peak demand, etc. On the other hand, real-time analysis would concentrate on analyzing a small window of traffic data with a view to provide a quick and possibly dirty warning of Impending/ongoing traffic anomalies. Real-time analysis may rely on less sophisticated analysis because of the resource demands and imminence of attacks.

II EXISTING METHODOLOGY

General properties of network packet traffic have been studies intensely for many years - standard references include. Many different analysis techniques have been employed in these and other studies including wavelets in. The majority of these traffic analysis studies have been focused on the typical, packet level and end-to-end behavior (a notable exception being). Our focus is mainly at the *flow level* and on identifying frequency characteristics of anomalous network traffic. There have been many prior studies of network fault detection methods. Example include. Feather *et al.* use statistical deviations from normal traffic behavior to identify faults while a method of identifying faults by applying thresholds in time series models of network traffic is developed in. These studies focus on accurate detection of deviations from normal behavior. Our work is focused on identifying anomalies by removing first from the signal its predictable, ambient part, and only then employing statistical methods. Wavelet are used for the former task. Detection of black-hat activity including denial-of-service(DoS) attacks and port scan attacks has also been treated widely. Methods for detecting intrusions include clustering , neural networks and Markov models. Moore *et al.* show that flow data can be effective for identifying DoS attacks . A number of intrusion detection tools have been developed in recent years in response to the rise in black-hat activity. An example is Bro which provides an extensible environment for identifying intrusion and attack activity. Our work complements this work by providing another means for identifying a variety of anomalous behaviors including attacks. We identify flash crowds as an important anomaly category. The events of September 11, 2001 and the inability of most online news services to deal with the offered demand is the most extreme example of this kind of behavior. While infrastructure such as content delivery networks (CDNs) have been developed to mitigate the impact of flash crowds, almost no studies of their characteristics exist. A recent study on flash crowds is by Jung. That work considers flash crowds (and DoS attacks) from the perspective of Web servers logs whereas ours is focused on network traffic. Finally, *cooperative pushback* is proposed in as a means for detection and control of events such as flash crowds.

III PROPOSED METHODOLOGY

In this project we are going to detect the anomalies using the following three techniques.

- Traffic Analysis at the Source
- General mechanism of detector.
- Trace.

Traffic Analysis at the Source:

We focus on analyzing the traffic at an egress router. Monitoring traffic at a source network enables early detection of attacks, to control hijacking of AD (administrative domain, e.g., campus) machines, and to limit the squandering of resources.

There are two kinds of filtering based on traffic controlling point as shown in. Ingress filtering protects the flow of traffic entering into an internal network under administrative control. Ingress filtering is typically performed through firewall or IDS rules to control inbound traffic originated from the public Internet. On the other hand, egress filtering controls the flow of traffic leaving the administered network. Thus, internal machines are typically the origin of this outbound traffic in view of an egress filter. As a result, the filtering is performed at the campus edge. Outbound filtering has been advocated for limiting the possibility of address spoofing, i.e., to make sure
That source addresses correspond to the designated addresses for the campus. With such filtering in place, we can focus on destination addresses and port numbers of the outgoing traffic for analysis purposes.

General mechanism of detector:

The first step is a traffic parser, in which the correlation signal is generated from packet header traces or NetFlow records as input. The first step is a traffic parser, in which the correlation signal is generated from packet header traces or NetFlow records as input. Fields in the packet header, such as destination addresses and port numbers, and traffic volume depending on the nature of the traffic, can be used as a signal. By this way we generate the signal.

Second step is to transform the signal using the discrete wavelet transform DWT. Analyzing discrete domains such as address spaces and port Numbers poses interesting problems for wavelet analysis. We employ the correlation in different domains to generate the suitable signal for analysis.

.

Trace:

To verify the validity of our approach, we run our algorithm on four traces of network traffic. First, we examine our method on traces from the University of Southern California that contain real network attacks. Second, to inspect the performance of our detector on backbone links, we examine the mechanism on KREONet2 traces, which include over 230 organizations, from July 21, 2003, to July 28, 2003, that contain real worm attacks . In the trace employed, there were three major attacks and a few instantaneous probe attacks, which were judged by various forensic traffic analyses in advance. Third, to compare our method with Snort, we exploit a live network in Texas A&M University. Fourth, to evaluate the sensitivity of our detector's performance over attacks of various configurations, we employ the attack-free traces from the NLANR (National Laboratory for Applied Network Research) , which are later superimposed with simulated virtual attacks.

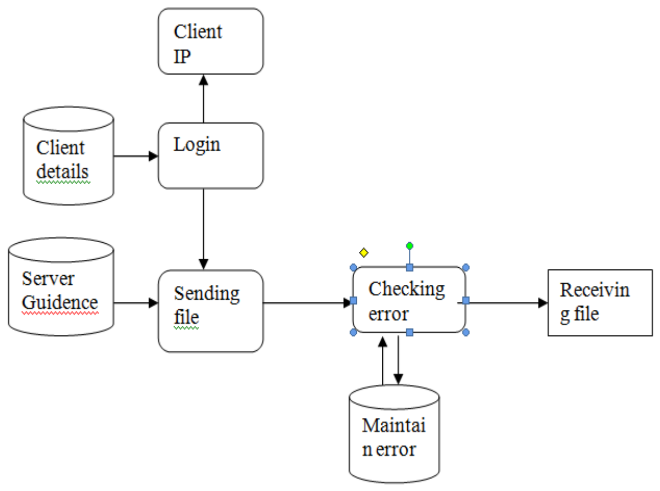


Fig 1: System Architecture

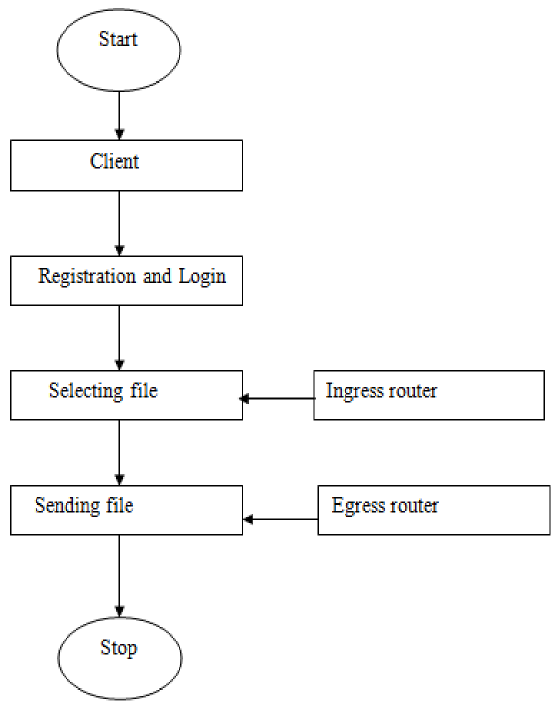


Fig 2: Module diagram

IV CODING

CLIENT

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Data.SqlClient;
using System.Threading;


private void btnSend_Click(object sender, EventArgs e)
    {
        if (txtFilename.Text != "")
        {
            lblError.Text = "Idle";

            try
            {
                //chk.Server = "192.0.0.12";
                //IPAddress[] ipAddress =
Dns.GetHostAddresses(chk.Server);
                IPAddress[] ipAddress =
Dns.GetHostAddresses(Servername);
                //IPAddress[] ipAddress =
Dns.GetHostAddresses("localhost");
                IPEndPoint ipEnd = new IPEndPoint(ipAddress[0],
5656);
               // double ip = chk.Server;
                //IPEndPoint ipEnd = new IPEndPoint(ip,5656);
                Socket clientSock = new
Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.IP);


                string filePath = "";


                strfile = strfile.Replace("\\", "/");
                while (strfile.IndexOf("/") > -1)
                {
                    filePath += strfile.Substring(0, strfile.IndexOf("/")
+ 1);
                    strfile = strfile.Substring(strfile.IndexOf("/") + 1);
                }


                byte[] fileNameByte =
Encoding.ASCII.GetBytes(strfile);
                if (fileNameByte.Length > 850*1024)
                //if (size > 850)
                {


                    lblError.Text = "File transferred."

                }
               catch (Exception ex)
               {
                if (ex.Message == "No connection could be made
because the target machine actively refused it")
                {
                    lblError.Text = "";
                    lblError.Text = "File Sending fail. Because server not
running.";
                }
                else
                {
                    lblError.Text = "";
                    lblError.Text = "File Sending fail." + ex.Message;
                }
            }

        }
        else
        {
            MessageBox.Show("Select a file to send");
        lblError.Text = "";
        lblError.Text = "Select a valid file";
    }
}

    private void sendFile_Load(object sender, EventArgs e)
    {

        btnAvail.Enabled = true;
    }

        private void btnAvail_Click(object sender, EventArgs e)
    {
        //string Filename = "v.txt";
        //string Servername = "spiro12";
        CheckServer objCheck = new CheckServer();
        objCheck.Show();
        this.Hide();
        string path = "\\\\" + Servername + "/DOTNET/" + Filename +
"";
        if (System.IO.File.Exists(path))
        {
            btnSend.Enabled = true;
        }

    }

    private void sendFile_FormClosed(object sender,
FormClosedEventArgs e)
    {
        delete();
        //string Servername = "spiro12";
        //string uname = "uname.txt";
        string path4 = "\\\\" + Servername + "/DOTNET/" + Filename +
"";
        FileInfo fi4 = new FileInfo(path4);
        File.Delete(path4);
        login log = new login();
        log.Close();
    }
    public void delete()
    {
        string fileName = "filename.txt";
        //string Servername = "Spiro12";
        string path1 = "\\\\" + Servername + "/DOTNET/" + fileName +
"";
        string filesize = "filesize.txt";
        string path2 = "\\\\" + Servername + "/DOTNET/" + filesize +
"";
          string server = "server.txt";
          string path3 = "\\\\" + Servername + "/DOTNET/" + server
+ "";

          FileInfo fi1 = new FileInfo(path1);
          File.Delete(path1);
          FileInfo fi2 = new FileInfo(path2);
          File.Delete(path2);
          FileInfo fi3 = new FileInfo(path3);
          File.Delete(path3);

        }
    }
```

Server

```
public class Server
    {
        SqlConnection cn = new SqlConnection("server=.;initial
catalog=Yokesh;uid=sa;pwd=;");

        SqlCommand cmd;
        SqlDataReader dr;
        IPEndPoint ipEnd;
        Socket sock;
        string servernameclient;
      string  filesizeclient;
        string filenameclient;
        string username;
        string strexe = "";
        public Server()
        {
          ipEnd = new IPEndPoint(IPAddress.Any, 5656);
          sock = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.IP);
          sock.Bind(ipEnd);
        }
      public static string receivedPath;
        public static string curMsg = "Stopped";
        public  void StartServer()
        {
          IAsyncResult ar=null ;
          while (true)
          {
            try
            {
              curMsg = "Starting...";
              sock.Listen(100);

              curMsg = "Running and waiting to receive file.";
              Socket clientSock = sock.Accept();

              byte[] clientDataa = new byte[1024 * 5000];

              int receivedBytesLen =
clientSock.Receive(clientDataa);


              curMsg = "Receiving data...";

              int fileNameLen = BitConverter.ToInt32(clientDataa,
0);
              string fileName =
Encoding.ASCII.GetString(clientDataa, 4, fileNameLen);
                 detection();
                 strexe = fileName.Substring(fileName.IndexOf(".") + 1);

                 //Thread.Sleep(10000);
                 //detection();
                 //strexe = fileName.Substring(fileName.IndexOf(".") + 1);

                 if ( Convert.ToDouble( filesizeclient) > 500 ||
strexe=="exe")
                 {
                   cn.Open();
                   cmd = new SqlCommand("insert into errorlog values
(@server,@uname,@filen,@files,@errorcount)", cn);
                   cmd.Parameters.Add(new SqlParameter("@server",
SqlDbType.VarChar));
                   cmd.Parameters.Add(new SqlParameter("@uname",
SqlDbType.VarChar));
                   cmd.Parameters.Add(new SqlParameter("@filen",
SqlDbType.VarChar));
                   cmd.Parameters.Add(new SqlParameter("@files",
SqlDbType.VarChar  ));
                   cmd.Parameters.Add(new SqlParameter("@errorcount",
SqlDbType.Int));
                   cmd.Parameters["@server"].Value = servernameclient;
                   cmd.Parameters["@uname"].Value = username;
                   cmd.Parameters["@filen"].Value = filenameclient;
                   cmd.Parameters["@files"].Value = filesizeclient;
                   cmd.Parameters["@errorcount"].Value = 1;
                   cmd.ExecuteNonQuery();
                   cn.Close();
                   //sock.EndAccept(ar);
                   //sock.EndReceive(ar);

                   //break;
                 }
                 else
                 {
                   cn.Open();
                   cmd = new SqlCommand("insert into errorlog values
(@server,@uname,@filen,@files,@errorcount)", cn);
                   cmd.Parameters.Add(new SqlParameter("@server",
SqlDbType.VarChar));
                   cmd.Parameters.Add(new SqlParameter("@uname",
SqlDbType.VarChar));
                   cmd.Parameters.Add(new SqlParameter("@filen",
SqlDbType.VarChar));
                   cmd.Parameters.Add(new SqlParameter("@files",
SqlDbType.VarChar));
                   cmd.Parameters.Add(new SqlParameter("@errorcount",
SqlDbType.Int));
                   cmd.Parameters["@server"].Value = servernameclient;
                   cmd.Parameters["@uname"].Value = username;
                   cmd.Parameters["@filen"].Value = filenameclient;
                   cmd.Parameters["@files"].Value =filesizeclient;
                   cmd.Parameters["@errorcount"].Value = 0;
                   cmd.ExecuteNonQuery();
                   cn.Close();
                   //sock.EndAccept(ar);
                   //sock.EndReceive(ar);

                 }
```

IX. OUTPUT SCREEN SHOTS

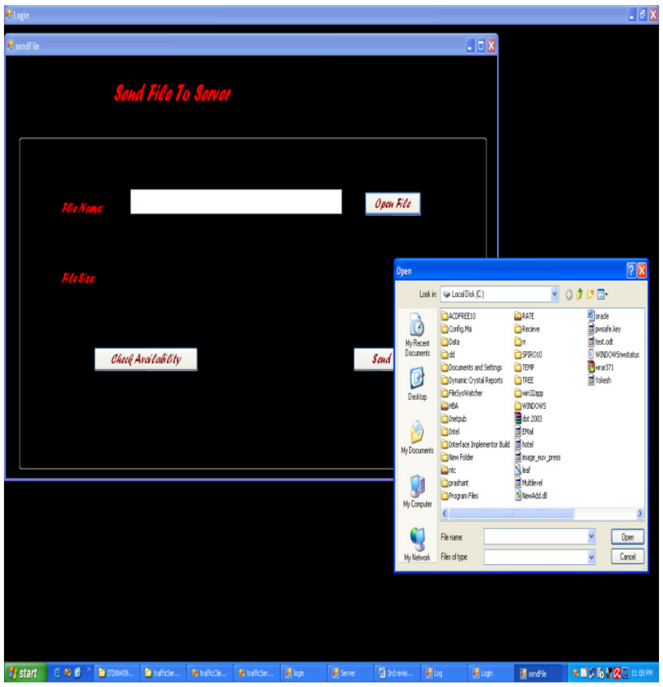
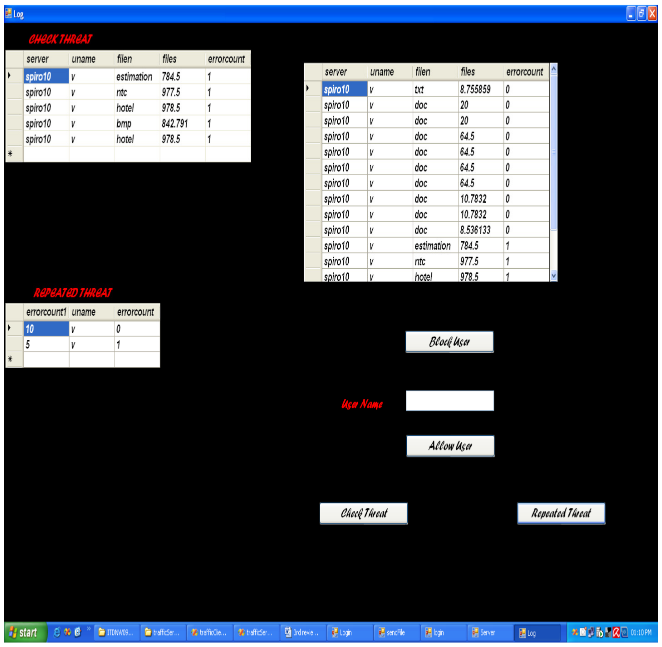


Fig 3: Send file to server



server	uname	filen	files	errorcount
spiro10	v	estimation	784.5	1
spiro10	v	ntc	977.5	1
spiro10	v	hotel	978.5	1
spiro10	v	bmp	842.791	1
spiro10	v	hotel	978.5	1

server	uname	filen	files	errorcount
spiro10	v	txt	8.755859	0
spiro10	v	doc	20	0
spiro10	v	doc	20	0
spiro10	v	doc	64.5	0
spiro10	v	doc	64.5	0
spiro10	v	doc	64.5	0
spiro10	v	doc	64.5	0
spiro10	v	doc	10.7832	0
spiro10	v	doc	10.7832	0
spiro10	v	doc	8.536133	0
spiro10	v	estimation	784.5	1
spiro10	v	ntc	977.5	1
spiro10	v	hotel	978.5	1

errorcount1	uname	errorcount
10	v	0
5	v	1

Fig 5: Check thread

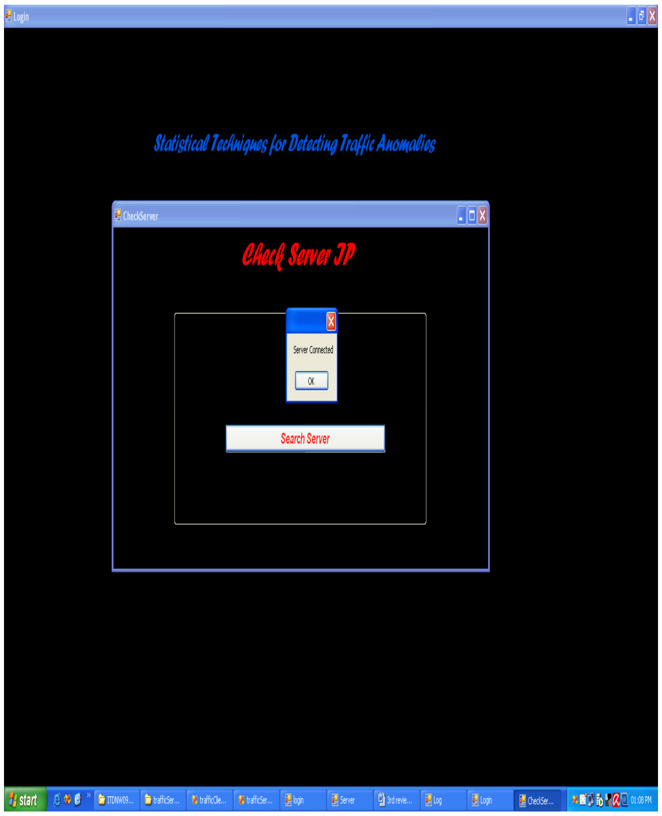


Fig 4: Check server IP

CONCLUSION

Finally we use the technique of finding the attack or the anomalies. This is done with the help of setting the threshold and we are comparing the result with the historical data and the anomalies are detected using the statically analysis. We report on our results employing correlation of destination addresses, port numbers and the distribution of the number of flows as monitored traffic signals.

REFERENCES

[1] Seong Soo Kim and A. L. Narasimha Reddy, Senior Member, IEEE “Statistical Techniques for Detecting Traffic Anomalies Through Packet Header Data “IEEE/ACM transactions on networking, vol. 16, no. 3, june 2008

[2] A. Ramanathan, “WADeS: A tool for distributed denial of service attack detection” M.S. thesis, TAMU-ECE-2002-02, Aug. 2002. .

[3] P. Barford *et al.*, “A signal analysis of network traffic anomalies,” in *ACM SIGCOMM Internet Measurement Workshop*, Nov. 2002..

[4] J. Mirkovic, G. Prier, and P. Reiher, “Attacking DoS at the source,” in *IEEE Int. Conf. Network Protocols*, Nov. 2002.

[5] C. Estan and G. Varghese, “New directions in traffic measurement and accounting,” in ACM SIGCOMM, Aug. 2002.

[6] A. Medina et al., “Traffic matrix estimation: Existing techniques and new directions,” in ACM SIGCOMM, Aug. 2002.

[7] I. H. Witten, A. Moffat, and T. C. Bell, Managing Gigabytes—Compressing and Indexing Documents and Images, 2nd ed. San Mateo, CA: Morgan Kaufmann, 1999, pp. 129–141.

[8] S. Mallat, “A theory for multiresolution signal decomposition: The wavelet representation,” IEEE Trans. Pattern Anal. Machine Intell., vol. 11, no. 7, pp. 674–693, 1989.

[9] A. Lakhina, M. Crovella, and C. Diot, “Diagnosing network-wide traffic anomalies,” in ACM SIGCOMM, Sep. 2004.

[10] M. Roesch, “Snort—lightweight intusion detection for networks,” in USENIX LISA 1999, Seattle, WA, Nov. 1999.