The Front Line Battle Against P2P

Douglas Ennis
Ringling School of Art and Design
2700 North Tamiami Trail
Sarasota, Florida 34234
941-309-4706
dennis@ringling.edu

Divyangi Anchan
Ringling School of Art and Design
2700 North Tamiami Trail
Sarasota, Florida 34234
941-309-4150
danchan@ringling.edu

Mahmoud Pegah
Ringling School of Art and Design
2700 North Tamiami Trail
Sarasota, Florida 34234
941-359-7625
mpegah@ringling.edu

ABSTRACT

One of the prevalent topics under discussion in educational environments today is the use of P2P (peer-to-peer) software. Aside from being major bandwidth-eaters, the use of such software to distribute and download copyrighted material has significant legal and financial implications for the campus community. Having a clear and concise policy restricting such use is one matter; enforcing said policy has proved to be an entirely different and complex task. Although it is elementary to prevent P2P traffic by blocking well-known ports used by such software, the vast majorities majority of P2P software, such as BitTorrent, download and upload pieces of files from different sources on different ports[?]. P2P protocols and client programs are working around 'issues' that prevent them from functioning and are becoming more and more 'intelligent'. The detection of P2P traffic based on well-known ports and 'port guessing' do not match the intelligence in newer generation P2P protocols and software. False positives generated with the use of well-known ports for detection are another serious concern. A good approach would be a solution that detects P2P and other undesirable traffic based on packet payload, as opposed to source and destination ports. Commercial products available to address these issues are expensive and not as flexible as the solution we discuss in this paper.

Our proposed model is a combination of several open-source solutions to which changes have been made to suit our environment and requirements. We also discuss other possible model solutions and discuss the pros and cons of such solutions.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection – Access controls, Information flow controls and Invasive software.

General Terms: Documentation, Performance, Security.

Keywords: Peer-to-peer software, Firewall, Router, Intrusion Detection, Log Server, Log parsing tool, Snort, Snortsam, IPTables.

1. INTRODUCTION

File sharing—the utilization of P2P (peer-to-peer) communication networks to disseminate computer files among users of the P2P communication networks—is one of the most important challenges for educational institutions to address. These new network protocols have significantly changed the way copyrighted materials, including digital music, video games, and digital videos can be distributed. Availability of fast network services on college campuses along with other resources such as CD/DVD production tools have made large scale distribution of copyrighted materials possible and easy. *"As of July 2002, KaZaA -- the most popular peer-to-peer (P2P) file-sharing network by far -- boasted 100 million registered users. By May 2003, KaZaA had become the worlds most downloaded software program of any kind, with 230.3 million downloads. All told, millions of users download over 2.6 billion copyrighted files (mostly sound recordings) each month via various peer-to-peer networks."*[10]

The widespread illegal distribution of copyrighted materials using well-known P2P applications is a concern not only for copyright owners but also for network administrators on college campuses where the file-sharing applications run. Ringling School of Art and Design, like any other higher education institution, had to grapple with the problems caused by P2P software, such as: reduced bandwidth, fiscal ramifications, compromised security, and potential legal backlash. These risks, combined with the increasing sophistication of P2P protocols, have made addressing this issue complex and require multiple approaches. At Ringling School of Art and Design we have found that the first line of defense must be education of the student body through partnerships. Partnering with the Board of Trustees, Student Life, President's Office, faculty, and the students made our policy clear and the expectations pronounced. Our policy was Zero-Tolerance and the students' expectations were the removal of their network services with the support of the president, student life, and campus faculty. We found that communication of our policy and the reasoning behind such a policy had a significant impact on students' use of P2P. Our "firm approach" for the less obliging student is an elaborate detection, logging, and enforcement system that made our policy something palpable. These systems detect the use of P2P protocols on our student housing and campus networks, and through analysis, logging, and the eventual restriction of network services. This "firm approach" combined with an honest smile and straightforward policy has significantly reduced the use of P2P on our campus. Within a short time of our system and policy integration, network traffic dropped by 80% and we ceased receiving notices from the RIAA and other similar agencies.

1.1 Partnerships

One of the first meetings between the Information Technology division and students regarding their usage of P2P software unearthed a major problem; students were unaware of the campus security, copyright infringement, and ethical use policy, or were unsure as to exactly what it encompassed. Our policy is signed by all students at the beginning of the school year and outlines acceptable and unacceptable usage of network resources and potential ramifications of breaking the policy. Hindsight reveals that the first week of school is not an appropriate platform to outline the usage of school resources let alone be the only interaction with said policy. Combating this gap in communication became our department's primary goal. Educating students on what is acceptable usage of campus resources and why these policies are in place was an effort that IT could not overcome by itself; the entire campus community had to embrace and be informed of the security, copyright infringement, and ethical use policy. Our first step was to receive the endorsement from the Board of Trustees and the President's Office to eliminate the sharing of copyrighted materials from our campus. Once we received this endorsement, the next step was to educate the Dean of Faculty, the faculty, and the Dean of Student Life on why these protocols are treacherous to our campus and how they can help. Some faculty have taken it upon themselves to include a statement concerning our firm stand and policy on P2P in their syllabus and communicate that they are partnering with the IT department to address and resolve the P2P issues on campus.

1.2 Partnerships Moving Forward

In the next year, IT plans on enhancing our communication with students and faculty in key ways. Our first initiative is to partner with the students from the beginning of the school year and continue our message throughout the year. In collaboration with the Director of Library Services, IT intends to hold seminars concerning copyright issues through the year. Discussions will encompass copyright laws and Ringling School of Art and Design's policy, why it is in place, and how this policy is protecting them. The second initiative is to enhance our partnership with Student Life and faculty, enabling them to talk with students directly and answer question about our campus security and standards of conduct in this information age. The communication message must be that the School is taking a stand against copyright infringement, not that the "IT department has taken away my KaZaA or Bittorrent." The key message must be the unity of the School, from the Board of Trustees, officers, senior administrators, school counselors, faculty, and staff.

2. POLICY ENFORCEMENT FRAMEWORK

Intrusion Detection System (IDS) hold a very important place in today's Information Technology (IT) security infrastructure. The use of IDS to detect impending or successful attacks has become commonplace; today IDS are taking on a new role, that of organizational policy enforcement entity aids. Most IDS use the principle of 'attack signatures' to identify attack traffic on the Network. This principle is now extended to detect traffic in violation of organizational policy.

Several commercial and open-source solutions are available for policy enforcement. Implementation of commercial products in educational organizations is often wrought with budgetary restrictions. Several open-source solutions are available that implement parts of the functionalities desired in a policy enforcement framework. While most organizations do implement the different network security related detection, prevention, and implementation entities, it has been our goal to integrate these entities into a consolidated environment where information can flow seamlessly between our firewalls, IDS, and log server, and can be viewed from a centralized console. Implementation of a centralized log server and its integration with the policy enforcement entities has been one of the primary goals in this project. In this section, we present the different building blocks that are part of our policy enforcement framework. All pieces of this framework are open-source based solutions, some of which are implemented without customization and others with source code modification and addition.

2.1 Background Information

Due to the limitation on the length of this paper, we are unable to provide detailed introductory material on the protocols, technologies, and open-source products referred to in this paper. It is beyond the scope of this paper to discuss installation instructions for each product referenced in this paper, and readers will be referred to installation instructions where appropriate. Interested users should refer to Snort [1], SnortSam [2], Syslog-ng [3] and IPTables [4] for more information.

Table 1 provides a summary of the implementation details of the building blocks of the framework.

Framework Entity	**Product/ Operating System**
IDS	Snort / Linux
Firewall	IPTables / Linux
Policy Enforcement Plug-in	SnortSam / Linux
Log Server	Syslog-ng / Linux
Log parser	Custom Perl Script
Front End	Custom Perl script and C programs

Table 1 Framework Entity Summary

2.2 Model Setup

A model of our policy enforcement framework is illustrated in Figure 1. The basic functionality of detecting policy infringements is performed by the implemented IDS. In our case, the IDS is Snort. The IDS, upon infringement detection, communicates with a policy implementation plug-in, which is capable of limiting access to the offending computer systems. This limitation can be in the form of blocking all further traffic from that source. The policy implementation plug-in in our framework is SnortSam, a program that resides in the firewall (IPTables) system and modifies the firewall rule-set based on directions received from the IDS. The plug-in sends log messages pertaining to changes made in the firewall rule-set to a centralized log server (Syslog-ng). A log parsing utility parses policy enforcement related logs on the log server. The presence of a centralized log server and a log parsing utility enables administrators to access such logs and other historical and aggregated information from a web-based front end.

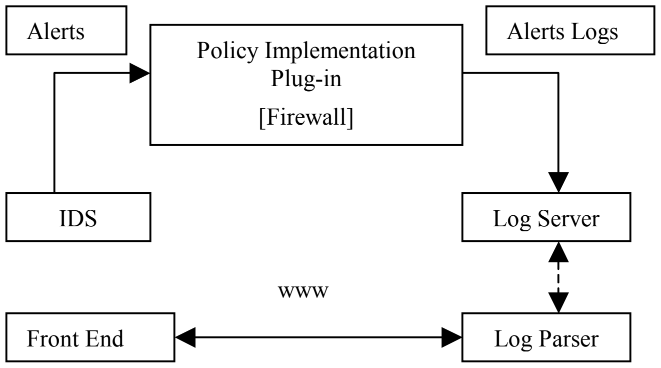

Figure 1 Model Policy Enforcement Framework

3. BUILDING BLOCKS

In this section, we discuss the different building blocks of the policy enforcement framework.

3.1 Intrusion Detection System - Snort

Snort [1] is a popular, open-source IDS written by Marty Roesch. It will not surprise us if many of the readers of this paper are Snort users. Basic installation and configuration instructions for Snort are packaged with the Snort distribution; the user manual can be found on the web [5]. Before Snort is compiled and installed, users should patch the Snort source to include the SnortSam output plug-in capabilities [6]. Instructions for applying the patch are available in the download.

3.2 Policy Implementation Plug-in - SnortSam

Snortsam [2] is a Snort plug-in developed by Frank Knobbs. This plug-in works in tandem with Snort to block IP addresses that trigger policy infringement alerts in Snort. SnortSam is currently implemented to work with a wide array of firewall products including IPTables, Cisco's PIX firewall, Cisco's popular Access Control Lists (ACL) on routers, Netscreen and Checkpoint firewall.

Although SnortSam is an excellent product in its basic implementation, the authors of this paper desired more than the simplistic approach of dropping offending traffic. IT shops strive to reduce support calls to their help desk staff ,and blindly blocking offending computer systems would have been a guaranteed way of increasing support calls. With simple modifications to the SnortSam source code, we introduced re-direction rules for IPTables which re-directed all traffic from the offending computer system to a website which displayed essential information such as reasons for network access shutdown, contact information to our IT department, etc., to the user. The reader can refer to Appendix A, Part I for source code corresponding to this section.

The logging framework for centralized logging was designed and implemented in-house from scratch. The default implementation of SnortSam logs to a flat file. SnortSam provides three default log levels: 0=off, 1=sparse, 2=normal, 3=verbose. We introduced a new log level: 4=log server. Source code modifications were made to recognize log server related configurations in the SnortSam configuration file and to process and direct log messages to the centralized log server when required. Source code was also added to call the logging function when IP address blocking and un-blocking rules were called in SnortSam. Readers can refer to Appendix A, Part II for information on source code corresponding to this section. The major source code modifications made to achieve the centralized logging functionality are summarized below:

1. Introduce log server related configuration options and a new log level to indicate logging to the log server (*snortsam.c*).
2. Extend the logging function (*logmessage()* in *snortsam.c*) in SnortSam to direct log messages to the log server based on the log level.
3. Invoke logging function (logmessage()) with log level = 4 for block and un-block rule changes made to firewall (call logmessage() in *ssp_iptables.c* for block and *snortsam.c* for un-block).

Readers can contact the authors of this paper for the modified version of SnortSam.

3.3 Log Server – Syslog-ng

Because of its enhanced feature-set, we decided to use Syslog-ng as the centralized log server. Installation instructions can be found at [3].

3.4 Log Parsing Utility

Log parsing and its presentation to a front-end for viewing are done in two parts.

The first part is a generic log parsing and log mailing utility (*logparser.pl*) that is capable of parsing logs from any source on the log server. Although there are several log parsing, evaluating, and report generating open-source solutions available [8], we found LogSentry [7] to be an excellent parsing utility which works in tandem with Syslog-ng and its logging mechanisms. Due to specific internal requirements and LogSentry's lack of mailing functionality, we have implemented a custom log parsing and log mailing utility (*logparser.pl*) in Perl. This tool is written on the lines of LogSentry and follows similar parsing logic as LogSentry. '*logparser.pl*' uses keywords in four files to weed out logs.

- logcheck.violations: contains all keywords that indicate a possibility of violations in log files. This file can be used to parse out policy violations using keywords 'BLOCKINFO' and 'UNBLOCKINFO' introduced in logs sent to the log server from the SnortSam plug-in for IPTables.
- logcheck.violations.ignore: is used to indicate keywords that generate false positives. '*logparser.pl*' ignores all logs that contain these keywords.
- logcheck.hacking: contains keywords that indicate intrusion attempts on the server generating the logs.
- logcheck.ignore: contain generic keywords that should be ignored in all log files.

'l*ogparser.pl'* outputs the above logs to files with name format 'type.output@*server name*' (where type=violation/ hacking). All other logs are output to a file named 'unusual.output@*server name*'.

The second part is a utility (blockUpdater.pl) that uses the parsed logs generated by *logparser.pl.* This utility pairs the different 'BLOCK' and 'UNBLOCK' logs based on their '*blockid*'s, and isolates IP addresses that are currently in 'block' stage. It then formats the logs for presentation to a web browser. This utility can be used alongside CGI (Common Gateway Interface) programs or other program written to interact with a web browser.
Readers can contact the authors of this paper to obtain these tools.

4. ACKNOWLEDGMENTS

We thank Dr. Larry Thompson, President, Ringling School of Art and Design, for his continuous support, strong advocacy for ethical use of information technology resources, and insightful vision for promoting the standards of information age conduct on campus. Kathleen List, the Director of Library Services, has educated us and our campus community on issues related to the copyright acts and issues; we are lucky to have her on board. We gratefully acknowledge the support of our faculty and students for partnering with us to address P2P issues on campus. Of course, this work would not be possible without the diligent work and support of our fellow staff at the Information Technology division.

5. REFERENCES

[1] *Snort*, http://www.snort.org

[2] *SnortSam*, http://www.snortsam.net

[3] *Syslog-ng*, http://www.balabit.com/products/syslog_ng

[4] *Netfilter / IPTables,* http://www.netfilter.org

[5] *Snort User Manual*, http://www.snort.org/docs

[6] *Snort patch for SnortSam*, http://www.snortsam.net/download.html

[7] *Linux 2.4 NAT Howto,* http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.html

[8] *Log Analysis and Parsing,* http://loganalysis.org

[9] *Logcheck/LogSentry, Log Parsing Utility,* http://sourceforge.net/projects/sentrytools

[10] *President of Recording Industry Association of America, Carl Sherman's testimony before Senate Committee on Commerce, Science, and Transportation, September 17, 2003.* http://commerce.senate.gov/hearings/testimony.cfm?id=919&wit_id=2584

6. APPENDIX A

Unless stated otherwise, the following code fragments should be added to the referenced source files. Text in bold format indicates additions made to existing code/ data structures. The authors' comments are in bold and italics. Code in close proximity to added code is included.

6.1 Part I

The SnortSam plug-in for IPTables is *'ssp_iptables.c'*. The default behavior of the plug-in is to add 'DROP' rules 'to the 'FORWARD' and 'INPUT' chains of IPTables. We replace the 'DROP' rules with Network Address Translation (NAT) rules. Readers must ensure that IPTables has been configured to load the modules required to perform NAT'ing on the firewall (the basic NAT module is: iptable_nat.o). Refer to [7] for information on NAT in IPTables.

Logging Before Modified Block Code (File: ssp_iptables.c)

```
Function void IPTBlock(BLOCKINFO*,void*) {
    struct tm *tp;
    <snip>
    if(bd->block) {
        snprintf(msg,sizeof(msg)-1,"Info: Blocking ip %s",
        inettoa(bd->blockip));
        logmessage(3,msg,"iptables",0);
        tp = localtime(&(bd->blocktime));
        snprintf(msg,sizeof(msg)-1,"BLOCKINFO:%d: Blocking
                access to and from %s on %04i/%02i/%02i at
                %02i:%02i:%02i for %d  seconds",
                bd->blockid,inettoa(bd->blockip),
                tp->tm_year+1900,tp->tm_mon+1,tp->tm_mday,
                tp->tm_hour,tp->tm_min,tp->tm_sec,
                bd->duration);
        logmessage(4,msg,"iptables",bd->blockip);
    <snip>
}
```

Modified Block Code:

```
/* Assemble command */
 if (snprintf(iptcmd,sizeof(iptcmd)-1,
"/sbin/iptables -t nat -A PREROUTING -p tcp -s %s -j DNAT
--to-destination w.x.y.z:80", inettoa(bd->blockip)) >= sizeof(iptcmd)) {
            snprintf(msg,sizeof(msg)-1,"Error:  Command  %s  is  to  long",
            iptcmd);
            logmessage(1,msg,"iptables",0);
            return;
}
if (snprintf(iptcmd2,sizeof(iptcmd2)-1,
"/sbin/iptables  -I  INPUT  -s  %s  -j  DROP",  inettoa(bd->blockip))  >=
sizeof(iptcmd2)) {
            snprintf(msg,sizeof(msg)-1,"Error:  Command2  %s  is  to  long",
            iptcmd2);
            logmessage(1,msg,"iptables",0);
            return;
}
```

Modified Un-Block Code:

```
/* Assemble command */
if (snprintf(iptcmd,sizeof(iptcmd)-1,
"/sbin/iptables -t nat -D PREROUTING -p tcp -s %s -j DNAT
--to-destination w.x.y.z:80", inettoa(bd->blockip)) >= sizeof(iptcmd)) {
            snprintf(msg,sizeof(msg)-1,"Error:  Command  %s  is  to  long",
            iptcmd);
            logmessage(1,msg,"iptables",0);
            return;
}
if (snprintf(iptcmd2,sizeof(iptcmd2)-1,
"/sbin/iptables  -D  INPUT  -s  %s  -j  DROP",  inettoa(bd->blockip))  >=
sizeof(iptcmd2)) {
            snprintf(msg,sizeof(msg)-1,"Error:  Command2  %s  is  to  long",
            iptcmd2);
            logmessage(1,msg,"iptables",0);
            return;
}
```

6.2 Part II

In this section we present the source code additions made to SnortSam to extend its logging functionality.

File: snortsam.h

```
# include <limits.h>
<snip>
/* Log Server Defaults */
#define LOGSERVER_PORT        514 /* Default syslog UDP Port */
#define MAX_LOGSERVERPORT 65535
<snip>
typedef struct _blockinfo { /* Block info structure */
                unsigned long blockip;
                unsigned long peerip;
                time_t duration;
                time_t blocktime;
                unsigned short port;
                unsigned short proto;
                unsigned short mode;
                short block;
                unsigned long blockid;
}               BLOCKINFO;
```

The variable 'blockid' helps pair the 'block' and 'un-block' log messages in the log server. The log parsing utility can use this variable to weed out IP addresses that were blocked and un-blocked and only display to the front-end IP addresses that are in the 'block' stage.

```
/* Enumerated data structure to indicate log server socket type */
enum socktype {TCP, UDP};
/* Log Server Structure */
typedef struct _logserver {
                struct sockaddr_in socklogserver;
                enum socktype stype;
                struct _logserver *next
}               LOGSERVER;
```

This structure is used to store log server related information including. A pointer is provided ('next') for future use to maintain a list of log servers.

File: snortsam.c

```
LOGSERVER *logserverp = NULL:
LOGSERVER mylogserver;
<snip>
Function void getout(int) {
     <snip>
     if(callersock)
                closesocket(callersock);
     if(mysock)
                closesocket(mysock);
     if (lssock)
                closesocket(lssock);
     <snip>
}
<snip>
Function void logmessage (unsigned int, char*, unsigned long) {
     int sendlen;
     <snip>
     if (!daemonized) {
                printf(logmsg);
                printf(“\n”);
     }
     if ((loglevel == 4) && (level == 4)) {
          printf("Sending logs to the logserver %s\n",
          inettoa(logserverp->socklogserver.sin_addr.s_addr));
          if (logserverp->stype == TCP) {
                if ((lssock = socket(AF_INET,SOCK_STREAM,0))==-1){
                snprintf(msg,sizeof(msg)-1,"Error creating log server
                socket!");
                logmessage(3,msg,"logmessage",0);
          }
     }
     else {
          if ((lssock = socket(AF_INET,SOCK_DGRAM,0)) == -1){
                snprintf(msg,sizeof(msg)-1," Error creating log server
                socket!");
                logmessage(3,msg,"logmessage",0);
          }
     }
     if(connect(lssock,
        (struct sockaddr*)&(logserverp->socklogserver),
         sizeof(struct sockaddr)) == -1) {
                snprintf(msg,sizeof(msg)-1,"Unable to contact log server.");
                logmessage(3,msg,"logmessage",0);
     }

     sendlen = strlen(logmsg);
     send(lssock,logmsg,sendlen,0);
     closesocket(lssock);
     <snip>
}

Function parseline ( char*, bool, char* unsigned long) {
     <snip>
     else if(!stricmp(arg,"loglevel {
        remspace(val);
        loglevel=atol(val);
     }
     else if (!stricmp(arg,"logserver")) {
        char* token;
        char* tailptr;
        unsigned long port;
        char tempval[STRBUFSIZE+2];
        remspace(val);
        strcpy(tempval,val);

        logserverp=&mylogserver;

        /* Fill the sockaddr_in struct */
       logserverp->socklogserver.sin_family=AF_INET;

       /* Look for port number */
       if (strstr(tempval,":") == NULL) { /* Single Value */
          if ((strstr(tempval,".")) == NULL) {
                snprintf(msg,sizeof(msg)-1,"Format for logserver is
                  IP:Port:Protocol. Atleast IP should be provided");
                logmessage(3,msg,"snortsam",0);
                exit(1);
          }
       else { /* Valid IP...hopefully */
          if((logserverp->socklogserver.sin_addr.s_addr =
                         getip(tempval)) == 0) {
                snprintf(msg,sizeof(msg)-1,"Error resolving log
                  server '%s',    val);
                logmessage(3,msg,"snortsam",0);
                exit(1);
          }
          snprintf(msg,sizeof(msg)-1,"Using Default Port
                %d\n",LOGSERVER_PORT);
          logmessage(3,msg,"snortsam",0);
          logserverp->socklogserver.sin_port =
                         htons(LOGSERVER_PORT);
       }
       else { /* More Values */
          token = strtok(tempval,":");
          if (( logserverp->socklogserver.sin_addr.s_addr =
                         getip(token)) == 0){
                 snprintf(msg,sizeof(msg)-1,"Error resolving log
                    server '%s'.", val);
                 logmessage(3,msg,"snortsam",0);
                 exit(1);
          }
          if ((token = strtok(NULL,":")) != NULL) {
                 port = strtoul(token,&tailptr,0);
                 if ((port == ULONG_MAX) || (port >=
                         MAX_LOGSERVERPORT)) {
                    snprintf(msg,sizeof(msg)-1,"Please specify valid
                         logserver port\n");
                    logmessage(3,msg,"snortsam",0);
                    exit(1);
                 }
                 else {
                    logserverp->socklogserver.sin_port =
                         htons(port);
                 }
          }
          else {
                 snprintf(msg,sizeof(msg)-1,"Using Default Port
                         %d\n",LOGSERVER_PORT);
                 logmessage(3,msg,"snortsam",0);
                 logserverp->socklogserver.sin_port =
                         htons(LOGSERVER_PORT);
```

```
            }
        }
        memset(&(logserverp->socklogserver.sin_zero),'\0',8);
        if (strstr(val,"TCP") != NULL) {
            logserverp->stype = TCP;
        }
        else if (strstr(val,"UDP") != NULL) {
            logserverp->stype = UDP;
        }
        else
            logserverp->stype = UDP;

        logserverp->next = NULL;
    }
    <snip>
}

Function block ( SENSORLIST*, unsigned long, unsigned short, unsigned
long, unsigned short, unsigned short, time_t, unsigned char, time_t) {
        <snip>
        /* checks here */
        if (dontblockhost(blockip)) {
            snprintf(msg,sizeof(msg)-1,"Ignoring block for white-listed
              host %s.",inettoa(blockip));
            logmessage(3,msg,"snortsam",snortbox->snortip.s_addr);
        }
        else {
            blockdata.blockip=blockip;
            /* Assign an ID for the request */
            blockdata.blockid = (unsigned long)rand();
        }
        <snip>
}

Function parsefile (char*, bool, char*, unsigned long) {
        <snip>
        fclose(fp);
        /* Ensure if loglevel = 4 then a logserver has been given. */
        if ((loglevel != 4) && (logserverp != NULL)) {
            snprintf(msg,sizeof(msg)-1,"Logserver configuration
                        ignored since loglevel is not 4");
            logmessage(3,msg,"snortsam",0);
        }
        if ((loglevel == 4) && (logserverp == NULL)) {
            snprintf(msg,sizeof(msg)-1,"A logserver configuration
                        is required when loglevel is 4");
            logmessage(3,msg,"snortsam",0);
            exit(1);
        }

}

Function main {
        <snip>
        /* Unblock Handler */
        <snip>
        time(&ttime);
        tp=localtime(&ttime);
        snprintf(msg,sizeof(msg),"UNBLOCKINFO:%d: Unblocking
               access to and from %s on %04i/%02i/%02i at
               %02i:%02i:%02i",bhp->blockinfo.blockid,
               inettoa(bhp->blockinfo.blockip),tp->tm_year+1900,
               tp->tm_mon+1,tp->tm_mday,tp->tm_hour,
               tp->tm_min,tp->tm_sec);
        logmessage(4,msg,"snortsam",0);
        addrequesttoqueue(FALSE,&(bhp->blockinfo),FALSE,FALSE);
                                   /* add unblock request to queue */
        <snip>
}
```