Global Information Assurance Certification Paper

Global Information Assurance Certification

GIAC Certified Intrusion Analyst

Mario R. Ricci
SANS Columbia, MD, USA
December 2002
GIAC GCIA Practical version 3.3
Submit date June 18, 2003

Abstract

This practical assignment is for the GIAC GCIA version 3.3. The practical assignment consists of three parts, a white paper, three analyses of a specific detect and an analysis of a weeks worth snort logs from a University.

The white paper provides a Snort preprocessor proof of concept. The Snort preprocessor exhibits a method of determining if the traffic that the sensor is picking up is normal by comparing the amount of unicast packets to broadcast packets in normal and different fault conditions.

The three analyses were of raw logs from the incidents.org web site. The first analysis dealt with crafted packets with 255.255.255.255 source IP address. The second analysis dealt with snort rule that triggers when based on content. The analysis showed the content was simply a jpeg and not an attack. The third analysis dealt with a NMAP scan that was using a decoy to obscure the real address and some of the problems associated with P2P software.

The week's worth of snort logs from a University showed active NIMDA within the network and a barrage of external scans. A detailed analysis was performed on all the alerts as well as an overall analysis with recommendations.

Table of Contents

Part 1: The State of Intrusion Detection

Detecting Configuration and Architecture Changes

Summary

The Intrusion Detection Systems (IDS) used to collect anomalous data from networks occasionally exhibit abnormal behavior due to configuration and architecture changes. The quality and quantity of the data collected by the sensor is compromised due to these configuration and architecture changes. The focus of this paper is to code a Snort preprocessor "proof of concept" to detect some common configuration and architecture changes. The results can be used to alert an analyst when some of the changes occur.

Introduction

Mr. Stephen Northcutt describes in his course material, "Intrusion Detection in Depth, Intrusion Detection Patterns " [Northcutt, 2002], a situation where his Computer Incident Response Team (CIRT) observed only echo responses without expected echo requests. At first it appeared to be an unauthorized back door. It was determined to be a false alarm when it was discovered that a switch configuration was changed. Similar configuration and architecture changes can occur that reduce the effectiveness of the sensor due to conditions external to the sensor software. An analyst could use a smarter sensor that would detect and alert when they occur.

This paper is comprised of four sections. The first section describes some of the common configuration and architecture changes that can affect the sensor's ability to detect traffic. The second section describes the creation of configuration and architecture changes to produce data for statistical analysis. The third section analyzes the captured data to determine if the configuration and architecture changes can be detected through scrutinizing the data stream. The fourth section describes the process I used in writing the Snort preprocessor to scrutinize the data and the code for the preprocessor.

Section I: Common Configuration and Architecture Changes

Change 1: Interface is Discriminating

For the IDS sensor to work properly the network interface card software must pass all traffic to the operating system. Network interface cards usually read all packets that are on the wire, but the interface card software discriminates by discarding all unicast packets that are not destined for the device. The sensor's interface used to collect the traffic must be placed in promiscuous mode prevent this from happening. In figure 1, the sensor's network interface card connected to the switch is not in promiscuous mode, thus the sensor will only receive packets destined for the sensor's interface ethernet address and ethernet broadcast packets.

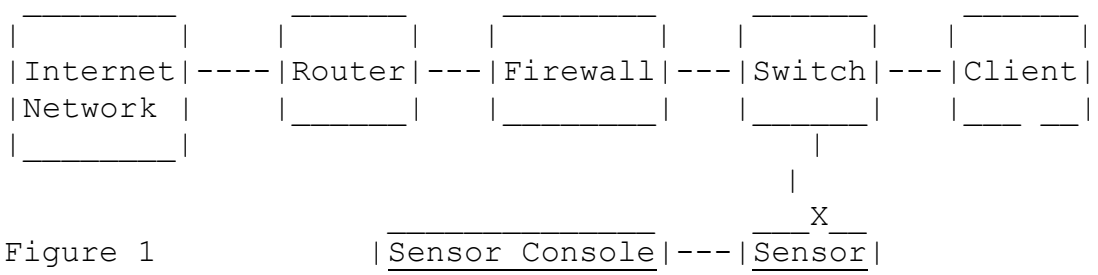


Figure 1

The administrator or IDS program not properly changing the interface to promiscuous mode or another program (tcpdump) turning promiscuous off when it exits can cause this to occur.

Change 2: Switch is Discriminating

The first Ethernet networks physical topology used a single cable (bus) that all devices connected to in series to participate on the network. All devices could read packets that were sent out from any device on the network. With the introduction of the star physical topology using twisted pair wiring, each device had its own wire connected to a central device know as a hub. The hub would repeat all signals received to all the devices connected. Intelligence was later built into the hubs so they learned what devices were connected to each port and only forwarded uni-cast data intended for the recipient. These intelligent hubs are also known as switches.

IDS sensors connected to switches require the switch to forward all packets to the sensor and not discriminate based on the ethernet destination address. Configuring the Switch Port Analyzer (SPAN) on Cisco switches allows data to be forwarded to the sensor as well as the intended node. In figure 2, the switch port connecting the sensor must be set to SPAN the switch in order for the sensor to receive all the traffic entering the switch. Similar to an interface card being in discriminating mode (non-promiscuous mode), if the port is not set to SPAN the switch, the sensor will only receive packets destined for the sensor's interface address and broadcast packets.

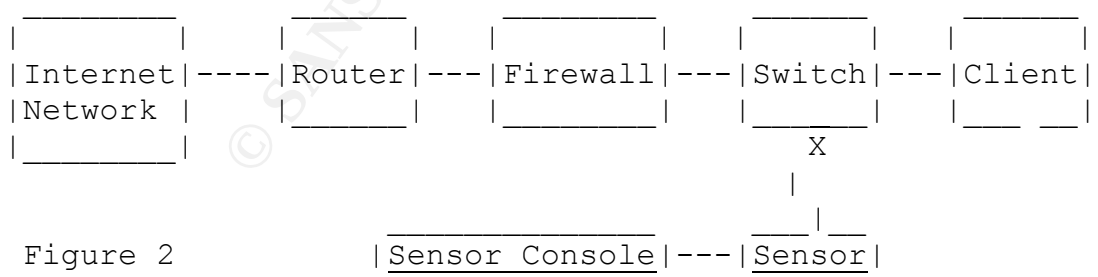


Figure 2

Configuring another port to SPAN or configuring the active switch configuration and not saving it to NVRAM are two possible causes for this to occur.

Change 3: Network ReRouting

IDS sensors are usually placed at ingress/egress points where traffic is aggregated. Aggregation points are usually designed to be fault tolerant. Automated fail over can include dynamic routing for rerouting of packets due to device failure. A sensor placed on the primary path may not capture the packets that are rerouted due to a network fault. Of course, an IDS sensor could be placed on secondary paths as well, but this is not always the case. In figure 3, a break has occurred between the lower router and the firewall. The data continues to follow through the upper router and firewall. The sensor could still receive some traffic or be completely isolated depending on how the fail over is configured.

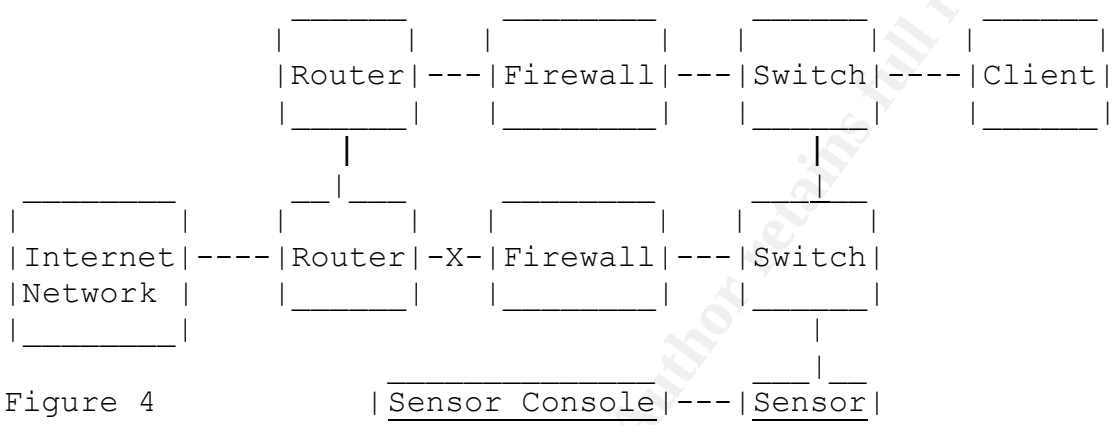


Figure 4

Network device failure and maintenance is a common cause of network traffic rerouting.

Section II: Creation of Configuration and Architecture Changes

To create an accurate representative data I used a live network. I was given permission to capture the data from a live network with the strict adherence to non-disclosure of the actual data, thus no logs are included. I was also restricted from creating a network rerouting change since this could impact the network.

The captures were performed using a laptop with Linux Redhat running tcpdump. To simulate the first change and capture a 1,000 packet baseline, the laptop was connected to a hub that was inserted between the production IDS sensor and switch. Tcpdump was run with the options as shown below to capture 1000 packets and stop (-c), capture link layer (-e), not convert protocol and port numbers (-nn) and write the captured packets to the file withprom (-w withprom).

```
tcpdump –c 1000 –e –nn –w withprom
```

Creating the first change was simple enough. Tcpdump was run with the same options as the basline capture and –p paramenter , to not put the card in promiscuous mode and of course a different file name to write the captured packets.

```
tcpdump –c 1000 –e –nn –w withoutprom -p
```

To simulate the second change I moved the laptop connection from the hub and connected it directly to a switch port of the switch that the hub was connected but was not configured to SPAN. Tcpdump was run with the same options as the baseline command as shown below with the exception of a different file name to write the captured packets.

```
tcpdump –c 1000 –e –nn –w withoutspan
```

Section III: Comparison and Analysis of Captured Data Streams

I compared the different samples to determine if there was a difference in the composition of the ethernet unicast to broadcast traffic. As shown in the table below, the ratio of unicast Ethernet packets to broadcast Ethernet packets clearly indicate the effect of these changed configurations on the traffic composition.

	Unicast	Broadcast	% of Unicast
Baseline count	1000	0	100%
Promiscuous disabled	841	159	84%
SPAN disabled	21	979	2.1%

A capture of a 1,000 packets is not representative of the complete network traffic pattern but enough to determine that the presence of even a five percentage of broadcasts is indicative of a configuration or architecture change. Comparing the amount of Ethernet broadcast to Ethernet unicasts can assist in detecting these changes.

Findings

The systems we use to collect data from the network configurations and architectures occasionally change. The composition of Ethernet unicast to Ethernet broadcast data collected by the sensor maybe altered due to these changes. The comparison of the broadcast ethernet traffic to unicast ethernet traffic can assist in identifying a changed state. This can be performed by taking a snapshot of the percentage of ethernet broadcast packets gained with simple combinations of commands or a more elaborate Perl scripts. A SNORT preprocessor can also be used to produce alerts when the broadcast percentage exceeds a certain threshold.

Section IV: The Snort Preprocessor

Abbreviated List of Steps

The comparison of network traffic composition can be performed in a Snort preprocessor since it can keep statistics of the many packets as they pass through the Snort program. The following is an abbreviated list of the steps I took to write the Snort Preprocessor.

1. Downloaded Snort 2.0 and Snort rules 2.0
2. Used the templates supplied with snort 2.0 to build the preprocessor.
3. Copied the `/home/snort/snort-2.0.0/src/template/spp_template.h` and `spp_template.c` to `/home/snort/snort-2.0.0/src/preprocessor/spp_ricci.h` and `spp_ricci.c`.
4. Changed the references from template to ricci in the files `spp_ricci.c` and `spp_ricci.h`.
5. Added the line `#include spp_ricci.h` in `/home/snort/snort-2.0.0/src/plugbase.h`
6. Added the line `SetupRicci();` in `/home/snort/snort-2.0.0/src/plugbase.c`.
7. Ran the *make* command from directory `/home/snort/snort-2.0.0`.
8. Ran the *make install* command from directory `/home/snort/snort-2.0.0`.
9. Commented out all lines in `spp_ricci.c` that produced compilation errors and changed the version number of Snort to show me that the `spp_ricci.c` preprocessor was actually being compiled and executed.
10. Created the subdirectory "log".
11. Ran the Snort command, *snort -A full -b -c /snortrules/rules2.0/rules/snort.conf -l log*
12. Added the spp_ricci preprocessor and commented all the rules and preprocessors in the snort.conf .

These steps were repeated over and over and over and over again.

13. Found similar code to use in preprocessor.
14. Found files that needed to be included.
15. Modified spp_ricci.c to perform desired actions.
16. Repeated `make` and `make install` commands until desired results were achieved.
17. Produced the following entries in the Snort alerts file.

```
[**] [1:1:1] broadcast exceeds thershold [**]
06/08-14:38:52.605492 192.168.0.2:3251 -> 192.168.1.100:1143
TCP TTL:128 TOS:0x0 ID:2848 IpLen:20 DgmLen:48 DF
******S* Seq: 0xA17B35FE  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
```

Overview of Snort Program

The program code logic for counting packets to determine the percentage of broadcast to unicast is fairly simple. The code for determining the percentage of broadcast packets is as follows.

This increaments the counter nNumpackets to give the total number of packets.

```
nNumPackets++;
```

This set the memory location bcast to contain the hexadecimal value FFFFFFFFFFFF

```
memset(bcast, 0xff, 6);
```

This compares the current packet ethernet address to the value of bcast. If they match, it increments the counter nNumBroadcastPackets to give the total number of broadcast packets.

```
if (memcmp((u_char *)p->eh->ether_dst, (u_char *)bcast, 6) == 0)
{
      nNumBroadcastPackets++;
}
```

This triggers the following code when the packet count reaches 100

```
if (nNumPackets == 100 )
{
```

This check is to see if the number of broadcasts equals zero. If so, it sets the percent to zero and skips the following division step. If not, the percent of broadcast is determined by dividing the number of packets by the number of Broadcast packets. This is necessary to avoid floating point divide by zero errors.

```
      if ( nNumBroadcastPackets == 0 )
      {
            nPercentBroadcastPackets = 0;
      }
      else
      {
            nPercentBroadcastPackets = nNumPackets/nNumBroadcastPackets;
      }
```

This reset the counters.

```
      nNumPackets = 0;
      nNumBroadcastPackets = 0;
```

This causes the alert to print when the percentage is greater than to equal to 25%.

```
      if (nPercentBroadcastPackets >= 25 )
      {
            snprintf(outstring, 255, "broadcast exceeds thershold");
            CallAlertFuncs(p, outstring, NULL, &event);
      }
}
```

The difficult part of writing the preprocessor for me was integrating the preprocessor with the Snort program. The acquiring of the packet data and the outputting to the alert file took some work on my part. Fortunately there was plenty of source code from other Snort preprocessor functions that I could review.

The acquiring of the packet data is done through the use of a pointer to the data structure that contains the packet data. A pointer is simply a variable that contains a memory address. This is useful in sharing data between programs since the entire packet data doesn't have to be transferred between programs. Another section of code that was used in the acquiring of the data was the data parsing performed in the Snort *decode* program. The *decode* program defined and loaded the data structure to make the data readily accessible by the preprocessor function. The code *p->eh->ether_dst* in the if statement *if (memcmp((u_char *)p->eh->ether_dst, (u_char *)bcast, 6) == 0)* represent the pointer to the packet data (*p*) the pointer to the ethernet header inside of the packet data(*eh*) and the pointer to the ethernet destination address inside the ethernet header(*ether_dst*).

The outputting of messages to the Snort alerts file is done through the Snort log program functions. The one I used is *CallAlertFuncs.* The Call Alert Function has to be passed four arguments. The first is the pointer to the current packet data structure. The second is a string of charters that will be displayed with the alert. The third can be a Null value. The forth must be the memory location of the event data.

CallAlertFuncs(p, outstring, NULL, &event);

The event data is produced with the *SetEvent function*. The function takes seven arguments. The first is the memory location of the event variable. The remaining six effects the way the alert is written to the alerts file.

SetEvent(&event, 1, 1, 1, 0, 0, 0);

The remainder of the code in the preprocessor program includes files that are required and creates variables. The code for the spp_ricci.c and spp_ricci.h files follows. The comments were deleted to save space

spp_ricci.h

```
/* $Id$ */
/* Snort Preprocessor Plugin Header File Ricci */

/* This file gets included in plugbase.h when it is integrated into the rest
 * of the program.  Sometime in The Future, I'll whip up a bad ass Perl script
 * to handle automatically loading all the required info into the plugbase.*
 * files.
 */
#include "snort.h"

#ifndef __SPP_RICCI_H__
#define __SPP_RICCI_H__
```

```
typedef struct _TemplateData
{
    /* your data goes here! */
} TemplateData;

/* list of function prototypes for this preprocessor */
void SetupRicci();
void RicciInit(u_char *);
void ParseRicciArgs(char *);
void PreprocRicci(Packet *);
/*void PreprocRestartFunction(int);
void PreprocCleanExitFunction(int);*/


#endif  /* __SPP_RICCI_H__ */
```

spp_ricci.c

```
/* $Id$ */
/* Snort Preprocessor Plugin Source File spp_ricci.c/
 *   by Mario Ricci <mricci20012002@yahoo.com>
 *   *   Version 0.0.1
 */

/* spp_ricci
 *
 * Purpose:
 *
 * The percentage of broadcast to unicast packets in the traffic stream
 * can be used to detect when a fault has occured on the support system.
 * When adding a plugin to the system, be sure to
 * add the "Setup" function to the InitPreprocessors() function call in
 * plugbase.c!
 *
*/
#include "spp_ricci.h"
#include "util.h"
#include "plugbase.h"
#include "debug.h"
#include "detect.h"
#include "log.h"

#include <stdarg.h>
#include <syslog.h>
#include <errno.h>
#include <sys/stat.h>
#include <time.h>
#include <signal.h>
#include <unistd.h>

int nNumPackets = 0;
int nNumBroadcastPackets = 0;
int nPercentBroadcastPackets = 0;
u_int8_t bcast[6];

/* external globals from rules.c */
extern char *file_name;
```

```
extern int file_line;


void SetupRicci()
{
    RegisterPreprocessor("ricci", RicciInit);
    fprintf(stderr, "\n-*> Setup Ricci! <*-\n");

    LogMessage("SetupRicci Function\n");
    DEBUG_WRAP(DebugMessage(DEBUG_PLUGIN,"Preprocessor: Ricci is setup...\n"));
}
void RicciInit(u_char *args)
{
   DEBUG_WRAP(DebugMessage(DEBUG_PLUGIN,"Preprocessor: Ricci Initialized\n"));
    ParseRicciArgs(args);
    LogMessage("RicciInit Function\n");
    AddFuncToPreprocList(PreprocRicci);
}

void PreprocRicci(Packet *p)
{
      nNumPackets++;
      memset(bcast, 0xff, 6);

      char outstring[255];
      Event event;
      snprintf(outstring, 255, "ricci write to alert file");
      /* fprintf(stderr,"outstring is %s:\n", outstring); */
      SetEvent(&event, 1, 1, 1, 0, 0, 0);

      if (memcmp((u_char *)p->eh->ether_dst, (u_char *)bcast, 6) == 0)
      {
            nNumBroadcastPackets++;
      }
      if (nNumPackets == 100 )
      {
            if ( nNumBroadcastPackets == 0 )
            {
                  nPercentBroadcastPackets = 0;
            }
            else
            {
                  nPercentBroadcastPackets = nNumPackets/nNumBroadcastPackets;
            }
            nNumPackets = 0;
            nNumBroadcastPackets = 0;
            if (nPercentBroadcastPackets >= 25 )
            {
                  snprintf(outstring, 255, "broadcast exceeds thershold");
                  CallAlertFuncs(p, outstring, NULL, &event);
            }
      }
}
```

References

Peter Jones September 2002 presentation on Promiscuous Mode and RPR
http://www.ieee802.org/17/documents/presentations/sep2002/pj_prom_03.pdf

http://www.cisco.com/univercd/cc/td/doc/product/lan/c2900xl/29_35xu/olhelp/spancfg.htm

Intrusion Detection in Depth – SANS 2002-2002, Stephen Northcutt, version 4.4

http://www.tcpdump.org/

http://www.snort.org/

Part 2: Network Detects

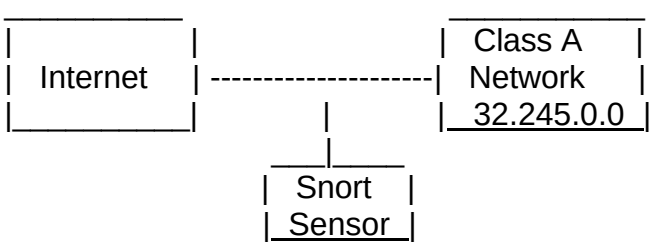
Detect #1: Broadcast from eleet

The detect was generated using tcpdump with the command **`tcpdump -v -r 2002.9.20 src host 255.255.255.255`**. The format of the detect is as follows

```
Time src > dst: flags data-seqno ack window urgent options

19:07:57.236507 255.255.255.255.31337 > 32.245.72.31.printer: R [bad
tcp cksum 1714!] 0:3(3) ack 0 win 0 [RST cko] (ttl 15, id 0, len 43,
bad cksum 2ea3!)

21:30:27.466507 255.255.255.255.31337 > 32.245.191.169.printer: R [bad
tcp cksum 1815!] 0:3(3) ack 0 win 0 [RST cko] (ttl 15, id 0, len 43,
bad cksum b617!)

21:54:39.476507 255.255.255.255.31337 > 32.245.47.26.printer: R [bad
tcp cksum 1714!] 0:3(3) ack 0 win 0 [RST cko] (ttl 15, id 0, len 43,
bad cksum 47a8!)

22:03:00.506507 255.255.255.255.31337 > 32.245.24.118.printer: R [bad
tcp cksum 1913!] 0:3(3) ack 0 win 0 [RST cko] (ttl 15, id 0, len 43,
bad cksum 5f4a!)

22:03:06.466507 255.255.255.255.31337 > 32.245.242.216.printer: R [bad
tcp cksum 1815!] 0:3(3) ack 0 win 0 [RST cko]  (ttl 15, id 0, len 43,
bad cksum 82e8!)

23:01:42.506507 255.255.255.255.31337 > 32.245.185.48.printer: R [bad
tcp cksum 1616!] 0:3(3) ack 0 win 0 [RST cko] (ttl 15, id 0, len 43,
bad cksum bb92!)

23:51:18.636507 255.255.255.255.31337 > 32.245.128.49.printer: R [bad
tcp cksum 1616!] 0:3(3) ack 0 win 0 [RST cko] (ttl 15, id 0, len 43,
bad cksum f491!)

00:28:54.706507 255.255.255.255.31337 > 32.245.79.69.printer: R [bad
tcp cksum 1714!] 0:3(3) ack 0 win 0 [RST cko] (ttl 15, id 0, len 43,
bad cksum 277d!)

02:14:30.246507 255.255.255.255.31337 > 32.245.75.84.printer: R [bad
tcp cksum 1714!] 0:3(3) ack 0 win 0 [RST cko] (ttl 15, id 0, len 43,
bad cksum 2b6e!)

02:22:24.226507 255.255.255.255.31337 > 32.245.11.174.printer: R [bad
tcp cksum 1913!] 0:3(3) ack 0 win 0 [RST cko] (ttl 15, id 0, len 43,
bad cksum 6c12!)

02:36:24.306507 255.255.255.255.31337 > 32.245.181.90.printer: R [bad
tcp cksum 1616!] 0:3(3) ack 0 win 0 [RST cko] (ttl 15, id 0, len 43,
bad cksum bf68!)
```

1. Source of Trace.

The source of the trace is a tcpdump binary file from Http://www.incidents.org.logs/Raw/2002.9.20. The actual network configuration was not provided. The sensor appears to be collecting information on packets destined for the class "A" network 32.0.0.0 subnet with a 16 bit mask, 32.245.0.0.



2. Detect was generated by.

The detect was generated by a Snort intrusion detection system. The ruleset was not provided. A modified default Snort rule set from http://www.snort.org/dl/rules/snort-stable.tar.gz was used. I modified the default snort.rules file by defining a home network of 32.245.0.0/16 and uncommented the backdoor.rules. Snort 1.9.0 processed the detect file with the modified rules. The file contained 12,455 packets of which all where TCP. Alerts were triggered on 12,072 packets. The alerts were comprised of 6002 scans on port 8080 (http proxy), 5999 scans on port 3128 (squid proxy), 25 alerts from the spp_portscan2 preprocessor, 6 MISC tiny fragments and 40 BACKDOOR Q access.

This analysis is on the BACKDOOR Q access alerts. The alerts were triggered because the snort rule BACKDOOR Q access looks for packets with a source IP address that begins with all 1's in the first three octets (255.255.255), anything in the last octet (0) and destined for the variable $HOME_NET which in this case is defined as any packets with an IP address that has 32. 245 in the first two octets and anything else on the last two octets (0.0). The rule is:

```
alert tcp 255.255.255.0/24 any -> $HOME_NET any (msg:"BACKDOOR Q access"; flags:A+; dsize:
>1;  reference:arachnids,203; sid:184; classtype:misc-activity; rev:3;)
```

The snort rule output is:

```
[**] [1:184:3] BACKDOOR Q access [**]
[Classification: Misc activity] [Priority: 3]
10/19-19:07:57.236507 255.255.255.255:31337 -> 32.245.72.31:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43 ***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
[Xref => arachnids 203]
```

3. Probability the source address was spoofed.

Definitely spoofed. All devices on the Internet require a unique address. When a device communicates with another it is through the use of a unique address or unicast. When a device wants to communicate with more than one device it uses a broadcast addresses. One

of these broadcast addresses has all the bits as 1. This equates to an address of 255.255.255.255. Since 255.255.255.255 is a reserved broadcast address it should never show up as a source address. The address is definitely spoofed. I confirmed this with W. Richards Stevens TCP/IP Illustrated Volume 1, page 45, figure 3.9.

IP address			Can appear as		Description
net ID	subnet ID	host ID-	source?	destination?	
-1		-1	**never**	OK	Limited broadcast never forwarded

-1 means a field of all one bits.

But why would someone send an illegal packet that they knew they would not get a response. Perhaps a raw IP stack triggers the targeted system to take an action when the 255.255.255.255 source address is received. This would prevent the target from accidentally being tripped since no one should use this address.

4. Description of attack.

This is a scan to send trojan programs a "cko" command. The action the trojan program will take is unknown. The signature is similar to the remote shell and admin tool Q. The snort rule reference to arachnids 203 is listed on the Whitehats webs site http://www.whitehats.com/ids/. The description is *"This indicates an attempt to send a command to a compromised Q server. Q is a backdoor that allows an attacker to run commands remotely as root, among other functions. "* The Whitehats web site CVE reference CAN-1999-0660 description is *"A hacker utility or Trojan Horse is installed on a system, e.g. NetBus, Back Orifice, Rootkit, etc."* There is no reference to Bugtraq or advICE.

The packet contains an invalid source IP address of 255.255.255.255. The source port of 31337 is well known for its use by hackers. The destination IP address is random. No pattern could be observed.

The destination port of 515 is used by the line printer daemon. Many networks allow for port 515 to enter the network with no authentication to support remote printing. It has no significance to the host since it should be discarded by the host when it is received.

The IP identification value is 0. A IP ID of 0 is allowed but it should be incremented each time a packet is sent out so each packet should have a different IP ID even retransmission would have a different IP ID.

The starting sequence number value, acknowledgement number value, and window size is 0. Most IP implementations will not use 0 for these values. The time to live (TTL) value of 15 in all packets indicates they originated from the same source since different sources would have different hop counts and result in different TTLs where the packet is received. This is a relatively low number so it is probably being crafted as well.

The ACK and RST flags are set in the packets. The combination of flags signals to immediately terminate the current session without a response. Since it has no current

session no action should take place on the targeted system. The RST flag would suppress any response from being sent as well.

5. Attack mechanism.

The attack works by having previously installed trojan programs listening for packets received with a source address of 255.255.255.255. The attack communicates to the trojan program through raw sockets similar to the Q program, from http://mixter.warrior2k.com/, although the attack was not generated using the Q program. This allows the attacker to fill the protocol header fields and listen for raw packets rather than pending on the operating system. The attack uses a less developed raw sockets program then the Q program since many of the values in the protocol fields are set to 0. This indicates the programmer didn't take the time to create the routines to fill the packets with random values to mimic actual TCP/IP values.

The attack is not targeting the destination service on port 515 (line printer daemon) since the packet will not make it through the protocol stack to the service since the reset flag is set. The use of port 515 could be a covert signal to the trojan program or selected as a possible open port into the network.

The attacker never intends to communicate through the TCP/IP protocol specifications. The combination of a spoofed IP address, using connection oriented protocol like TCP and setting the reset flag, all seem to show the originator doesn't expect to receive a typical packet in return.

The packets appear to be sent to random IP address. The time interval between the packets varies from a few seconds to hours. The short internal between the packets and volume of packets indicates an automated program is performing the probe as opposed to command line entries. The long interval between packets indicates that the target addresses of the program are larger than the subnet that is being monitored.

6. Correlations.

A clear consensus doesn't exist for this type of detect. I listed references that contradict as well as support my analysis.

Crist Clark contradicts my theory. He suggests the packet is from a broken worm or tool in his e-mail reply located at http://lists.insecure.org/lists/incidents/2001/Jul/0023.html

> *"The techniques you referenced all deal with sending packets to a broadcast or multicast _destination_ address. I am not aware of any useful attacks using the broadcast address, 255.255.255.255, as the SOURCE. I believe the general impression is that the 255.255.255.255:31337 to *:515 packets are the product of a broken worm or tool."*

Trent Riddel has another theory as well. He suggests the detect is an *"exploit that is looking for host that may have unpatched LPR holes and this traffic is coming from poorly configured*

attack tool or maybe even an attempt at a worm" in his GCIA practical submission located at http://www.giac.org/practical/Trenton_Riddell_GCIA.doc.

Les Gordon gives three possible scenarios. The third and *"fairly likely"* scenario supports my theory. His detailed analysis for his GCIA Practical Detect submission with the same signature in his located at http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00221.html

> *1) Somebody playing a prank, although given that this activity has been going on for quite some time (see correlations), this may not be very likely, unless this person or persons have way too much time on their hands, don't get bored quickly, and are very easily amused.*
>
> *2) Someone is trying to write some malicious software, perhaps a worm, but don't really know what they are doing. Again, given the period of time that this has been occurring, I think it's unlikely that someone may have been simply experimenting for so long without getting some kind of reward for their efforts. If it were a worm, the exploit mechanism would have to be something else entirely, as these packets are unlikely to be exploiting any vulnerability directly. In this case, these packets would have to be an artifact of something else that the worm was trying to achieve. A possibility suggested by Christ Clark (see correlations) back in July 2001 is that it's a worm that cannot bind to the interface properly and the "all ones" IP address is the result of that.... or perhaps it is a worm and they really meant to have the worm perform a limited broadcast on the local network to communicate with other compromised devices - but this is likely to be noticed at some point and I haven't found any such reports..*
>
> *3) Reconnaissance - specifically, a potentially covert means of communication to perhaps activate, or run a command on a back-door program, DDoS server etc (al la Q, TFN2K etc). This is in my opinion, fairly likely,but this detect is full of contradictions.*

A posting to GIAC GCIA Version 3.2 Practical Detect by krautt@cox.net located at http://cert.uni-stuttgart.de/archive/intrusions/2002/09/msg00112.html , suggests that it is a Q program that is generating the attack.

7. Evidence of active targeting.

The attack-timing interval varies from a few seconds to over an hour. This indicates that the attack isn't targeting the specific subnet 32.245.0.0. I suspect the entire 32.0.0.0 network is being scanned by the attack.

8. Severity.

The packets themselves pose no significant risk since they require a trojan program to be installed prior to any action taking place.

The severity is calculated using the formula:

Severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Criticality is rated at two since all systems on the network are suspect to being probed.

Lethality is rated at three since we don't know what would happen if a trojan program is found and the trojan probably has root access on the target.

System countermeasures rated at 1 since this is a large scan across many hosts that will have a percentage that are not properly secured and at risk to being infected by a trojan program.

Network countermeasures rated at 2 since the packets are obviously crafted and easily identified but allowed to pass. Network countermeasures currently in place are not evident from the detects and exact placement of the sensor is unknown.

Severity 2 = (2 + 3) – (1 + 2)

9. Defensive recommendations.

The most efficient recommendation is to stop these packets from entering the network. The use of an Access Control List (ACL) on a border router or rule in a firewall that discards source address of 255.255.255.255 would eliminate these packets from the network with little risk. A stateful inspection protocol would also deny these packets entry into the network.

Standard security practices should be implemented on the hosts to harden them. This includes installing all recommend patches when they are released, removing all unnecessary services, following system securing practices (i.e. shadow root, umask …) for the OS and anti-virus software with frequent updates to reduce the risk of infection by the trojan. Programs should also be prevented from being downloaded via the web and through e-mail attachments. Personal firewalls that detect and report unusual activity would be useful in the event a trojan is triggered and attempts to communicate to the master or attack another system. They could also discard the 255.255.255.255 packets when they reach the host.

10. Multiple choice test questions.

When should the address 255.255.255.255 be used as the source IP address?

```
a) When the source wants the destination to reply to all systems on
the local network.
b) When the source doesn't have an IP address and is using DHCP to
acquire one.
c) When the source is sending out a broadcast that it doesn't want a
reply to.
d) It should never be used.
```

```
Answer:d  The address 255.255.255.255 should never be used as a source
IP address.
```

11. Results of post to intrusions@incidents.org

Posted on February 10th and March 28th. Did not receive any responses to first to intrusions@incidents.org posting. Received three responses to second posting. Responses and replies are listed below.

11.1 Reply #1

Response

```
Date: Sat, 29 Mar 2003 19:30:37 +0100
From: "Andrew Rucker Jones" <arjones@simultan.dyndns.org>
To: "Mario Ricci" <mricci20012002@yahoo.com>
Subject: Re: Fwd: LOGS:GIAC GCIA Version 3.3 Practical Detect
```

Reply to response

Date: Fri, 4 Apr 2003 08:09:56 -0800 (PST)
From: "Mario Ricci" <mricci20012002@yahoo.com>
Subject: Re:LOGS:GIAC GCIA Version 3.3 Practical Detect
To: "Andrew Rucker Jones" <arjones@simultan.dyndns.org>
CC: intrusions@incidents.org

Andrew,

Thank you for your reply. I embedded my responses in italics after each of your comments.

Andrew Rucker Jones <arjones@simultan.dyndns.org> wrote:

Mario,
A very thorough analysis. I especially liked Your correlations section. This attack has been beaten to death by GCIA students, and You did a good job of finding a cross-section of their opinions.

> The actual network
> configuration was not provided. The sensor appears to be collecting
> information on packets destined for the class “A” network 32.0.0.0
> subnet with a 16 bit mask, 32.245.0.0.

How do You arrive at this conclusion and at the network diagram You included?

I used tcpdump to extract the source and destination addresses from all the packets. I then greped for common addresses within all the packets.

> the variable $HOME_NET which in this case is defined as any packets with
> an IP address that has 32. 24 in the first two octets and anything else
> on the last two octets (0.0).

Important typo here. "32.245".

Thank you for pointing this out. I will correct it in my analysis.

> This is a scan to send trojan programs a “cko” command.

...and what is a "cko" command?

I have not been able to determine what the "cko" will do. Assume that the command is for the trojan running on the target so it may have no relationship to a well known "cko" command.

> The use of port 515
> could be a covert signal to the trojan program or selected as a possible
> open port into the network.

How likely is it that this is a port that is open into the network?

I would say very likely based on my experience. The port is commonly used for access to LPD print servers. Print access is often over looked or minimized as a possible vector for attack. Encryption and authentication is often not available to print clients so the native port 515 has to be used.

> Network countermeasures rated at 1 since the packets are obviously
> crafted and easily identified but allowed to pass.

Are they really allowed to pass? This gets back to Your network diagram. Where in the target network is the IDS sensor?

The packets are allowed to pass at least to the point where the IDS sensor is installed. I am assuming the sensor is placed before any firewalls. Whether the packets are allowed to pass through the firewall cannot be determined.

11.2 Reply #2

Response

Date: Fri, 28 Mar 2003 14:44:29 –0700
From: Bryce_Alexander@vanguard.com
To: "Mario Ricci" <mricci20012002@yahoo.com>
Subject: Re: Fwd: LOGS:GIAC GCIA Version 3.3 Practical Detect

Reply to response

Date: Fri, 28 Mar 2003 17:50:56 -0800 (PST)
From: "Mario Ricci" <mricci20012002@yahoo.com
To: Bryce_Alexander@vanguard.com
Subject: Re: Fwd: LOGS:GIAC GCIA Version 3.3 Practical Detect

You say that it is not "Q" in your paper, but, you do not explain clearly why it isn't. Is this conclusion based on something someone said in some other link, or because you see the fields that are set to zero?

The fields being set to zero are one indication. The practical detect by Les Gordon listed in my correlations at http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00221.html explains in detail why it is not "Q".

thanks for the question.

Mario Ricci

11.3 Reply #3

Response

Date: Sat, 19 Apr 2003 20:02:35 -0400
From:"Carlos L. Edwards" <carlos.edwards@verizon.net>
To:"Mario Ricci" <mricci20012002@yahoo.com>
Subject: Re: Fwd: LOGS:GIAC GCIA Version 3.3 Practical Detect

Reply to response

Date: Wed, 23 Apr 2003 16:00:45 -0700 (PDT)
From:"Mario Ricci" <mricci20012002@yahoo.com>
To:"Carlos L. Edwards" <carlos.edwards@verizon.net>
Subject: Re: Fwd: LOGS:GIAC GCIA Version 3.3 Practical Detect

Mario,

You did an excellent job of presenting this detect with the ability to review it from multiple perspectives. It was interesting to hear what the analysis of others yielded.

Personally, I too would consider this as sleeper Trojan that Mr. Evil was attempting to awaken. Just out of curiosity, as various systems were targeted, did further analysis reveal any other suspicious activity at the conclusion of this possible "Q" scans. Just wondering if the remaining logs showed any activity that could have even remotely been the aftermath of this "Q" scan that successfully executed a Backdoor into one of the systems.

Let me know!

Carlos,

Thank you for the response. Unfortunately I only have access to alerts log. A review of the log showed 40 packets to all different address. None of these addresses trigger an alert as a source. Doesn't appear that a response was generated.

Detect #2: A visit to gay.com

The detect was generated using tcpdump with the command **`tcpdump –v –r 2002.9.28 host 64.125.138.190`** . The format of the detect is as follows

```
Time src > dst: flags data-seqno ack window urgent options
```

01:48:49.126507 32.245.166.236.64621 > 64.125.138.190.http: P 1801650240:1801653000(2760) ack 789190 win 33120 [tos 0x10]

02:00:20.226507 64.125.138.190.http > 32.245.166.236.61424: P 2639726813:2639728106(1293) ack 1476343 win 33120 (DF)

02:00:20.756507 64.125.138.190.http > 32.245.166.236.61425: P 1370563110:1370564114(1004) ack 1477309 win 33120 (DF)

02:00:20.876507 64.125.138.190.http > 32.245.166.236.61425: P 1242:2215(973) ack 641 win 33120 (DF)

02:00:21.046507 64.125.138.190.http > 32.245.166.236.61425: P 2454:3555(1101) ack 1281 win 33120 (DF)

02:00:21.456507 64.125.138.190.http > 32.245.166.236.61425: P
4775:5799(1024) ack 3176 win 33120 (DF)

02:00:21.586507 64.125.138.190.http > 32.245.166.236.61425: P
6038:7309(1271) ack 3816 win 33120 (DF)

02:00:21.696507 64.125.138.190.http > 32.245.166.236.61425: P
7548:8552(1004) ack 4456 win 33120 (DF)

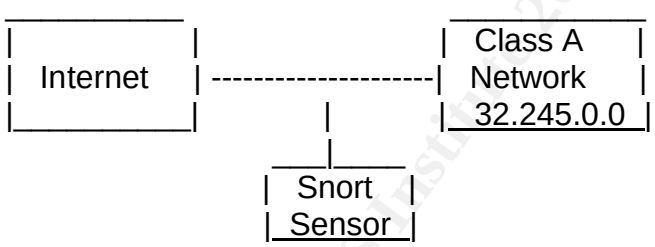
02:00:21.846507 64.125.138.190.http > 32.245.166.236.61425: P
8791:9818(1027) ack 5096 win 33120 (DF)

02:00:21.986507 64.125.138.190.http > 32.245.166.236.61425: P
10056:10682(626) ack 5735 win 33120 (DF)

02:00:22.136507 64.125.138.190.http > 32.245.166.236.61425: .
10921:12301(1380)
ack 6375 win 33120 (DF)

1. Source of Trace.

The source of the trace is a tcpdump binary file from Http://www.incidents.org.logs/Raw/2002.9.28. The actual network configuration was not provided. The sensor appears to be collecting information on packets destined for the class "A" network 32.0.0.0 subnet with a 16 bit mask, 32.245.0.0.



2. Detect was generated by.

The detect was generated by a Snort intrusion detection system. The rule set was not provided. A modified default Snort rule set from http://www.snort.org/dl/rules/snort-stable.tar.gz was used. I modified the default snort.rules file by defining a home network of 32.245.0.0/16 and uncommented all the include rules. I also removed the established flag for the flow parameter since the capture did not contain the complete session. Snort 1.9.0 processed the detect file with the modified rules. The file contained 4,864 packets of which 4863 where TCP and 1 UDP. Alerts were triggered on 650 packets. The alerts were comprised of 1 "Potentially Bad Traffic", 4 "system call detected", 10 "Web Application Attack", 55 "Attempt Information Leak", 73 "access to a potentially vulnerable web application", 131 "Misc activity", and 375 "Executable code was detected".

This analysis focuses on the 375 “Executable code detected”. The alerts were triggered because the snort rule looks for packets with string of 0100 0011 or hex 43 that repeats at least 24 times within the payload. The rule is:

```
alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS
(msg:"SHELLCODE x86 inc ebx NOOP"; content:"|43 43 43 43 43 43 43 43
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43|";
classtype:shellcode-detect; sid:1390; rev:3;)
```

The snort rule output is:

```
[**] [1:1390:3] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/28-02:00:20.756507 64.125.138.190:80 -> 32.245.166.236:61425
TCP TTL:50 TOS:0x0 ID:37600 IpLen:20 DgmLen:1044 DF
***AP*** Seq: 0x51B12226  Ack: 0x168ABD  Win: 0x8160  TcpLen: 20
```

3. Probability the source address was spoofed.

The source doesn't appear to be spoofed. The packet is in response to a request for a web page from the site 64.125.138.190. The address 64.125.138.190 resolves to the web site www.gay.com.

4. Description of attack.

This is not an attack. This is a false positive. I visited the web site and was able to reproduce the data stream that triggered the alert. The 24 '43' pattern is contained in a file image encoded with JPEG Interchange File Format (JFIF) and compressed using the Joint Photographic Experts Group (JPEG) compression method. A captured packet follows.

2:22:16.154154 64.125.138.190.http > my.net.168.0.3.33238: P [tcp sum ok] 132533:133642(1109) ack 3962

```
win 32832 <nop,nop,timestamp 1988004643 4906010> (DF) (ttl 55, id 27813, len 1161)
0x0000   4500 0489 6ca5 4000 3706 46e3 407d 8abe      E...l.@.7.F.@}..
0x0010   c0a8 0003 0050 81d6 f220 7b8d 939d 466a      .....P.... {...Fj
0x0020   8018 8040 4c37 0000 0101 080a 767e 8b23      ...@L7......v~.#
0x0030   004a dc1a ffd8 ffe0 0010 4a46 4946 0001      .J........JFIF..
0x0040   0101 0048 0048 0000 ffdb 0043 000b 0708      ...H.H.....C....
0x0050   0a08 070b 0a09 0a0c 0c0b 0d10 1b12 100f      ................
0x0060   0f10 2118 1914 1b27 2329 2927 2326 252c      ..!....'#))'#&%,
0x0070   313f 352c 2e3b 2f25 2636 4a37 3b41 4346      1?5,.;/%&6J7;ACF
0x0080   4746 2a34 4d52 4c44 523f 4546 43ff db00      GF*4MRLDR?EFC...
0x0090   4301 0c0c 0c10 0e10 2012 1220 432d 262d      C.......... C-&-
0x00a0   4343 4343 4343 4343 4343 4343 4343 4343      CCCCCCCCCCCCCCCC
0x00b0   4343 4343 4343 4343 4343 4343 4343 4343      CCCCCCCCCCCCCCCC
0x00c0   4343 4343 4343 4343 4343 4343 4343 4343      CCCCCCCCCCCCCCCC
0x00d0   4343 ffc0 0011 0800 3900 3703 0111 0002      CC......9.7.....
```

5. Attack mechanism.

This is not an attack. This is a false positive. The objective of this type of attack is to insert commands into memory by sending more data then the target system allocated for the data. This results in the additional data being written into memory that another program was using. A problem with this type of attack is the hacker doesn't know where the system will start reading memory from. Fillers are sent to assist in having the evil code executed from the being and occupy more memory. These fillers are no operations commands or NOOP.

6. Correlations.

The snort web site lists no correlations for SID 1390 **http://www.snort.org/snort-db/sid.html?sid=1390**7. Evidence of active targeting.

No active targeting was taking place since this was not an attack.

8. Severity.

The packets pose no risk since this is a false positive. Since the scale is 1-5 the criticality and severity are rated at the lowest possible rating of 1.

The severity is calculated using the formula:

Severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Criticality is rated at 1 since the system appears to an end-user workstation that is visiting a web site.

Lethality is rated at 1 since this is not an attack.

System countermeasures rated at 3 since I am unable to determine what the system countermeasures are in place on the end-user workstation.
Network countermeasures rated at 4 since I am unable to determine what network countermeasures are in place with the exception that they were using an IDS system to monitor the traffic.

Severity -5 = (1 +1) – (3 + 4)

9. Defensive recommendations.

No recommendation since this is not an attack. Possible refinement of snort rules to reduce false positives.

10. Multiple choice test questions.

The combination of hex strings that snort rules often look for in the content of a packet can occur in normal traffic. To determine if the alert is a false positive .

a) Look up the source IP to determine where it is coming from.
b) Look to see if the source IP has sent similar packets to other devices within your network.
c) Look up the source address on web sites that list known attacker addresses.
d) Reconstruct the session to determine if it was normal traffic.
e) All of the above.

Answer: e) All of the above.

Detect #3: NMAP SCAN

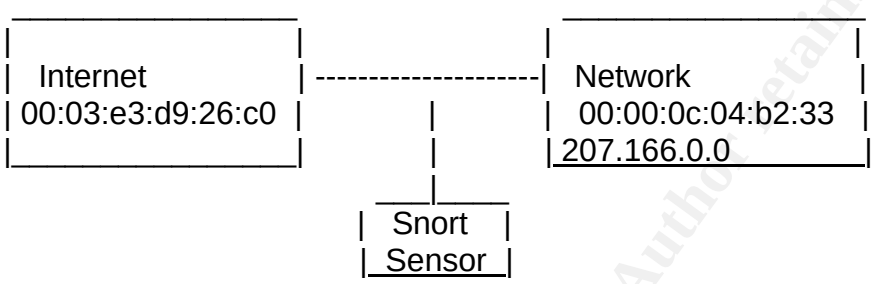
The detect was generated using snort with the command **`snort -r 2002.9.30 -l log -c /snortrules/rules/custom.conf –b`**. I then used the command **`cat log/alert | grep nmap -A5 > nmapalerts`** to extract all the NMAP alerts from log/alert file. I then deleted packets that didn't have a destination port of 9511.

```
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/30-04:33:04.926507 67.36.84.5:80 -> 207.166.135.150:9511
TCP TTL:50 TOS:0x0 ID:364 IpLen:20 DgmLen:40
***A**** Seq: 0x12E  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => arachnids 28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/30-04:33:10.006507 67.36.84.5:80 -> 207.166.135.150:9511
TCP TTL:50 TOS:0x0 ID:1086 IpLen:20 DgmLen:40
***A**** Seq: 0x1A7  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => arachnids 28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/30-04:33:15.016507 198.150.73.5:80 -> 207.166.135.150:9511
TCP TTL:50 TOS:0x20 ID:1702 IpLen:20 DgmLen:40
***A**** Seq: 0x214  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => arachnids 28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/30-04:33:20.036507 198.150.73.5:80 -> 207.166.135.150:9511
TCP TTL:50 TOS:0x20 ID:2442 IpLen:20 DgmLen:40
***A**** Seq: 0x294  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => arachnids 28]
--
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/30-06:24:16.096507 67.36.84.5:80 -> 207.166.135.150:9511
TCP TTL:50 TOS:0x0 ID:25402 IpLen:20 DgmLen:40
***A**** Seq: 0x276  Ack: 0x0  Win: 0x578  TcpLen: 20
```

[Xref => arachnids 28]

1. Source of Trace.

The source of the trace is a tcpdump binary file from Http://www.incidents.org.logs/Raw/2002.9.30. The file contains 15021 packets. The actual network configuration was not provided. The sensor appears to be collecting information on packets between two devices. This is based on only two Ethernet addresses were found in the trace (00:03:e3:d9:26:c0, 00:00:0c:04:b2:33). The source IP address of all packets generated from the 00:00:0c:04:b2:33 Ethernet address is 207.166.87.157. The destination IP addresses of all packets with the destination Ethernet address of 00:00:0c:04:b2:33 is 207.166.0.0 . The third octet addresses range from 207.166.3.0 to 207.166.252.0. It appears the monitored network consists of the network address 207.166.0.0 with a subnet mask of 255.255.0.0.



2. Detect was generated by.

This detect was generated by a Snort intrusion detection system. The rule set was not provided. A modified default Snort rule set from http://www.snort.org/dl/rules/snort-stable.tar.gz was used. I modified the default snort.rules file by defining a home network of 207.166.0.0/16 and uncommented all the include rules. I also removed the established flag for the flow parameter since the capture did not contain the complete session. Snort 1.9.0 processed the detect file with the modified rules. The file contained 15,020 packets of which all were TCP. Alerts were triggered on 11,095 packets and 11,066 were logged. The alerts were classified as 1 Potentially Bad Traffic, 19 Misc activity, and 11,046 Attempted Information Leak. The 11,046 attempted information leaks contained 27 SCAN NMAP TCP alerts. Of these 27 packets, 16 contained the destination port of 9511. This is unusual since this is not a normal port for services to be listening and thus unusual for a scan. This detect focuses on these 16 NMAP scan packet alerts. The rule is:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap
TCP";flags:A;ack:0; reference:arachnids,28; classtype:attempted-recon;
sid:628; rev:1;)
```

The snort rule output is:

```
[Classification: Attempted Information Leak] [Priority: 2]
10/30-18:24:24.186507 198.150.73.5:80 -> 207.166.135.150:9511
```

```
TCP TTL:50 TOS:0x20 ID:13404 IpLen:20 DgmLen:40
***A**** Seq: 0x3B2  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => arachnids 28]
```

3. Probability the source address was spoofed.

One of source IP addresses appears to be spoofed. The NMAP scan usually requires a response to collect the desired information. NMAP offers options to attempt to hide the real source address by using decoy source addresses and idle scans. The decoy source address packets are sent interspersed with the real source address. This makes it difficult to determine which was the actual source address that was scanning. The source addresses are from 2 different devices. The different source addresses along with source port, destination address and destination port are as follows:

```
 8 packets from IP address 67.36.84.5 port 80 to 207.166.135.150 port 9511
8 packets from IP address 198.150.73.5 port 80 to 207.166.135.150 port 9511
```

The following timing interval and TTL of the packets indicates that both packets are coming from the same device.

```
05:33:04.926507 67.36.84.5.http > 207.166.135.150.9511: ttl 50
05:33:10.006507 67.36.84.5.http > 207.166.135.150.9511: ttl 50
05:33:15.016507 198.150.73.5.http > 207.166.135.150.9511: ttl 50
05:33:20.036507 198.150.73.5.http > 207.166.135.150.9511: ttl 50

07:24:16.096507 67.36.84.5.http > 207.166.135.150.9511: ttl 50
07:24:21.136507 67.36.84.5.http > 207.166.135.150.9511: ttl 50
07:24:26.046507 198.150.73.5.http > 207.166.135.150.9511: ttl 50
07:24:31.136507 198.150.73.5.http > 207.166.135.150.9511: ttl 50

18:40:59.026507 67.36.84.5.http > 207.166.135.150.9511: ttl 50
18:41:04.036507 67.36.84.5.http > 207.166.135.150.9511: ttl 50
18:41:09.076507 198.150.73.5.http > 207.166.135.150.9511: ttl 50
18:41:14.086507 198.150.73.5.http > 207.166.135.150.9511: ttl 50

19:24:09.016507 67.36.84.5.http > 207.166.135.150.9511: ttl 50
19:24:14.136507 67.36.84.5.http > 207.166.135.150.9511: ttl 50
19:24:19.156507 198.150.73.5.http > 207.166.135.150.9511: ttl 50
19:24:24.186507 198.150.73.5.http > 207.166.135.150.9511: ttl 50
```

The source addresses are registered as follows.

```
67.36.84.5       Concordia University , 2701 Alma, Plano, TX
198.150.73.5     WiscNet, Madison, WI
```

Both addresses respond to a ping but neither of them is listening on port 80. I cannot determine which of the addresses is the decoy and which is the real one. This could be done by pinging from where the detect was made and comparing the TTL to the captured packets.

4. Description of attack.

NMAP is a scan to collect information about devices on a network. It is an opensource program found at insecure.org. It can be used to determine what devices are on a network, what ports are listening and what OS a system is running. It also attempts to determine if a firewall is blocking packets. It runs on most operating systems. The packets are identified from the 0 ack in the packets and acknowledgement flag being set.

It appears the NMAP is running an ack scan to access the firewall with the decoy option targeting port 9511. The command could be as follows

Nmap -sA –D67.36.84.5,ME -g 9511.
Nmap -sA –DME,198.150.73.5,ME -g 9511.

The packets are sent with a source port of 80 since many firewalls will allow traffic from a web server into the network that have the flag bits set to appear to be an established session. This could also be used to asses if a stateful or packet filter firewall is being used.

The description for the CVE Candidate doesn't seem to match the packet. The CAN-1999-0523 references ICMP packets. These packets are all TCP packets.

5. Attack mechanism.

The NMAP attack is a TCP port scan. This particular attack is sending packets to a targeted host. The response or no response indicates if the system is up. All packets received by the host should generate a reset regardless if the port is up or not since the session has not been established. A no response means the system probably isn't up or it is being filtered. The NMAP scan could also be trying to determine the operating system by the characteristics of the reset packet. This is often done with the TTL received, the TCP options and order of TCP options.

The interesting part of this detect is the target host destination port number of 9511. Port 9511 is not registered with IANA. A look at other alerts with a destination address of 207.166.135.150 and destination port of 9511 showed 73 packets. A look at the contents of these packets show the 207.166.135.150 is being sent GNUTELLA response packets (PONG) to a GNUTELLA request sent out (PING) as shown below.

```
06:44:36.106507 148.63.240.90.2538 > 207.166.135.150.9511: P 1932253194:1932253518(324) win 8192
(DF)0x0000   4500 016c 8113 4000 7106 f5eb 943f f05a        E..l..@.q....?.Z
        0x0010   cfa6 8796 09ea 2527 732b d80a 0000 0000        ......%'s+......
        0x0020   5e08 2000 8f08 0000 474e 5554 454c 4c41        ^.......GNUTELLA
        0x0030   2043 4f4e 4e45 4354 2f30 2e36 0d0a 5573        .CONNECT/0.6..Us
        0x0040   6572 2d41 6765 6e74 3a20 4d6f 7270 6865        er-Agent:.Morphe
        0x0050   7573 2032 2e30 2e31 2e38 0d0a 582d 556c        us.2.0.1.8..X-Ul
        0x0060   7472 6170 6565 723a 2046 616c 7365 0d0a        trapeer:.False..
        0x0070   504f 4e47 2d43 4143 4849 4e47 3a20 302e        PONG-CACHING:.0.
        0x0080   310d 0a58 2d4d 592d 4144 4452 4553 533a        1..X-MY-ADDRESS:
```

```
0x0090   2031 3438 2e36 332e 3234 302e 3930 3a35      .148.63.240.90:5
0x00a0   3835 330d 0a58 2d54 7279 3a20 3231 332e      853..X-Try:.213.
0x00b0   3437 2e32 332e 3139 303a 3934 3630 2c32      47.23.190:9460,2
0x00c0   342e 3137 342e 3133 352e 3234 333a 3835      4.174.135.243:85
0x00d0   3632 2c31 3331 2e39 362e 3131 322e 3439      62,131.96.112.49
```

GNUTELLA is a peer to peer (P2P) sharing program. Many GNUTELLA clients are available that to share files such as Morpheus, Bearshare, Gnucleus, Limewire, Phex, Swapper and XoloX.

Although the GNUTELLA packets appear to be a GNUTELLA response to a GNUTULLA request, all the GNUTELLA packets are retransmitted. It appears something is blocking the response packets from reaching 207.166.135.150.

My analysis is the 207.166.135.150 is running GNUTELLA and listening on port 9511. It has advertised this to the P2P community. Several devices have tried to communicate with 207.166.135.150 but failed because of an intervening firewall device. Someone has picked up on the fact that 207.166.135.150 is listening on this port and is attempting reconnaissance from the IP address 67.36.84.5 or 198.150.73.5 for possible future exploits.

Allowing GNUTELLA is a risk to a network because it provides access to workstations. Some of the GNUTELLA clients also include spyware, adware and push 3rd party software to report on the activity of workstations. Vulnerabilities in the GNUTELLA, GNUTELLA clients and 3rd party software open the network up to attack. The transferring of files can also introduce viruses into the protected network.

6. Correlations.

The mechanism for how NMAP works as well as the different options was found at Insecure.org web page http://www.insecure.org/nmap/idlescan.htm

Other references found are as follows.
The snort reference http://www.snort.org/snort-db/sid.html?sid=628
Whitehats reference http://www.whitehats.com/info/IDS28
The description for the CVE Candidate doesn't seem to match the packet. The CAN-1999-0523 references ICMP packets. These packets are all TCP.
CVE http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0523
advICE http://www.iss.net/security_center/advice/Intrusions/200310/default.htm

A good reference for how GNUTELLA works can be found at
http://www.toadnode.com/site_how_toadnode_works.ilx

Toadnode is a client that uses GNUTELLA. An overview of GNUTELLA can be found at
http://www.sans.org/rr/threats/gnutella.php

7. Evidence of active targeting.

The host with the IP address 207.166.135.150 port 9511 appears to be targeted. No packets were captured scanning other devices on the network from the same source addresses.

8. Severity.

The severity is calculated using the formula:

Severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Criticality is rated at 1 because the system appears to be workstation. No web, DNS or SNMP requests destined for this device.

Lethality is rated at 2 since a NMAP scan is designed to collect information about device not actually attack or compromise them.

System countermeasures rated at 1 since the person running the software doesn't understand the inherent risk associated with running P2P software and computer security. Assuming this is a workstation and the user is not familiar with computer security, the system is not hardened.

Network countermeasures rated at 3 since IDS detected the NMAP scan as well as the GNUTELLA packets. The retransmitted GNUTELLA and NMAP packets indicate the packets were being blocked when they attempt to enter the nework but the GNUTELLA software is communicating with other systems . This indicates that a stateful firewall is being used disallow packets that don't have an established session.

Severity -3 = (1 + 2) – (1 + 4)

9. Defensive recommendations.

The network appears to be able to hinder the GNUTELLA communications. It is not totally effective in stopping it because the workstation was able to transmit the address and port. GNUTELLA is designed to bypass firewalls and other network security measures. The most effective method would be to eliminate the GNUTELLA software from the network. An acceptable use policy should be drafted to make the use of P2P software inappropriate. Systems using this software would be removed from the network or users found be disciplined. Depending on the environment, the workstations could be locked down to prevent the loading of unapproved software.

A stateful inspection firewall would also reduce the effectiveness of NMAP to recon the network.

10. Multiple choice test questions.

The NMAP contains options to attempt to hide or obscure the address system performing the scan. Which of the following NMAP options are available? .

```
a) -sI, Idle scan.  Sending packets with a third parties address that
    has a predictable IPID.
b) -b FTP bounce.  Using a FTP server that allows proxies to scan other
    systems.
c) -D decoy scan.  Sends spoofed packets along with scan packets to
    obscure true source.
d) All of the above.

Answer: d) All of the above.
```

Part 3: Analyze This

1. Executive summary

Since this document will be publicly available, the actual University name and network addresses will not be used to protect the identity of the University. Any references to this specific University will use the term UNIVERITY. Any occurrences of the actual UNIVERSITY addresses will have the first two octets changed to xxx.yyy.

The UNIVERSITY in cooperation with SANS, provides logs to SANS to be analyzed as part of the SANS certification process. The logs are produced from a Snort IDS system with a typical configuration. The Snort system produces three sets of logs for each day, the alert, the scan and oos. The alert log is generated using the Snort fast option logs minimal packet information when the packets match the rules criteria. The scan log is produced from Snort Preprocessor portscan2 when a high number attempts are made from the same source to different systems. Portscan2 also logs packets with unusual flag combinations. The oos file contains full packet captures (up to the Snort limit) of packets with unusual header combinations.

The UNIVERSITY network is under constant scan activity and directed attacks. The UNIVERSITY network is also attacking other external systems. File-sharing software (P2P) runs unchecked throughout the network using up bandwidth opening vectors for attack. The Snort IDS sensor is being overloaded with false positives and traffic that should be blocked.

Four local systems appear to be infected with the NIMDA virus and numerous other systems appear to be infected with the Red Worm virus. Immediate actions should be taken to contain and eradicate these viruses. Immediate actions should also be taken to avoid and mitigate the risk associate with being connected to the Internet.

Detailed recommendations are listed with each separate alert. A comprehensive description of long term recommended actions is listed under the Defense Recommendations.

A detailed security assessment of the UNIVERSITY network cannot be performed with just the output of Snort IDS system. Many assumptions are made throughout this analysis in regards to the network architecture, security policy, firewalls, router ACLs and such due to the lack of information provided. This should be taken as a piece of the overall network security picture.

2. List of files analyzed

The files for this analysis were downloaded from http://www.incidents.org/logs. Five consecutive days of files were downloaded. I selected the dates April 1st through April 5th. The scan and alert files names represent the date that the data was captured. The OOS file names do not represent the data within the file. Most of the OOS files contain data from the previous day. The OOS data for April 5th was contained in the file named

OOS_Report_2003_04_07_31826. The files were generated by Snort with a fairly standard rulebase. The alert files are the output of snort running with -A fast option. The scan files are the output of the Snort Portscan Preprocessor. The OOS files are the output of Snort logging.

April 1st, 2003	OOS_Report_2003_04_02_24924	alert.030401.gz	scans.030401.gz
April 2nd, 2003	OOS_Report_2003_04_03_9924	alert.030402.gz	scans.030402.gz
April 3rd, 2003	OOS_Report_2003_04_04_29217	alert.030403.gz	scans.030403.gz
April 4th, 2003	OOS_Report_2003_04_05_3459	alert.030404.gz	scans.030404.gz
April 5th, 2003	OOS_Report_2003_04_07_31826	alert.030405.gz	scans.030405.gz

3. Meaningful analysis

The University network appears to be a large network based on the analysis of the snort logs. The combined alerts and scans file contains entries from 915 University systems that generated packets, but 43,379 UNIVERSITY systems are being sent packets. The five days of the alerts, not including the scansfile, contained 237,366 entries. The five days of the scans contained 1,416,553 entries. That is an average of 47,473 alerts per day, 1,978 per hour, 32 per minute and 1 every 2 minutes. The scans has an average of 283,310 per day, 11,804 per hour, 196 per minute and 3 per second. This sheer volume of alerts, scans and oos logs makes it difficult to tease the hostile activity from the normal traffic.

The Snort sensor appears to be placed on a network segment that is receiving and sending packets to the Internet. Minimal filtering appears to be occurring on the external perimeter router, based on the amount of NETBIOS packets that are being received. A filtering device does appear to be block egress from the network based on the NETBIOS packets not being recorded from any internal device.

Based on the source port of an entry with the UNIVERSITY source IP address, the UNIVERSITY had the following active services. These services were verified by accessing them. Other services were found using DNS references and Sam Spade.

Http on port 80

xxx.yyy.6.7	http://www.gl.UNIVERSITY.edu/
xxx.yyy.24.34	http://www.UNIVERSITY.edu/
xxx.yyy.100.165	linuxserver1.cs.UNIVERSITY.EDU
xxx.yyy.222.110	resnet2-348.resnet.UNIVERSITY.edu

https on port 443

xxx.yyy.24.20	https://listproc.UNIVERSITY.edu/lpUNIVERSITY/
xxx.yyy.24.33	https://my.UNIVERSITY.edu/fcgi-bin/myUNIVERSITY.fcgi

POP3 on port 110

xxx.yyy.100.230	mailserver-ng.cs.UNIVERSITY.edu v7.59

TFTP on port 69

xxx.yyy.105.48	aciv316pc-2.UNIVERSITY.EDU

SNMP on port 25

xxx.yyy.24.21	mx1in.UNIVERSITY.EDU sam spade
xxx.yyy.24.23	mx3in.UNIVERSITY.edu

xxx.yyy.6.34	mx3del.UNIVERSITY.EDU sam spade
xxx.yyy.6.35	mx2del.UNIVERSITY.EDU sam spade
xxx.yyy.6.40	mx4del.UNIVERSITY.EDU sam spade
xxx.yyy.6.47	mx1del.UNIVERSITY.EDU sam spade

DNS on port 53

xxx.yyy.1.2	UNIVERSITY2.UNIVERSITY.EDU sam spade
xxx.yyy.1.3	UNIVERSITY3.UNIVERSITY.EDU
xxx.yyy.1.4	UNIVERSITY4.UNIVERSITY.EDU sam spade
xxx.yyy.1.5	UNIVERSITY5.UNIVERSITY.EDU sam spade

Summary of import findings from detects.

Four internal systems, xxx.yyy.97.88, xxx.yyy.184, xxx.yyy.98.157 and xxx.yyy.97.66, appear to be infected with the NIMDA virus. Other systems may be compromised as well but not enough activity is recorded to substantiate this conclusion. The aggressive nature in which NIMDA spreads makes it a high probability that other systems within the UNIVERSITY are infected. A review of the content of these systems should be performed as well as actions to contain and eradicate the virus from the network.

Evidience ot the Red worm virus is also present on 66 internal systems. Of these 66, 9 are involved in other activity. There are address are as follows,

xxx.yyy.222.110, xxx.yyy.91.108, xxx.yyy.210.130, xxx.yyy.242.42,
xxx.yyy.210.222 xxx.yyy.87.70, xxx.yyy.233.78, xxx.yyy.206.74, xxx.yyy.163.135

A review of the content of these systems should be performed as well as actions to contain and eradicate the virus from the network.

An internal system is spoofing with Micosoft's IP address and sending packets out to the external network.

Egress and ingress filter should be performed to reduce the amount of alerts by removing the traffic from the network. The alerts should be left in place in the event the filtering should fail. Examples of this would be to drop all SMB packets, anti-spoofing and bad packet combinations.

The IDS senosrs are not tuned properly. The oos file contains may entries that are using the new ECN protocol and should not be log. The oos file also contains many entries from the Gnutella P2P networking software with illegal flag combinations that could be dropped rather than logged.

The alerts file contains a summary of the scans file and the oos file. The oos file also contains packets with bad flag combinations and many entires from Gnulleta. The oos external source addresses were not involved in significant attacks logged in the alerts as noted in the later analysis process section. The Gnutella traffic that is not an attack but does consume bandwidth and make another vector for act.

4. List of detects, priority by severity or number of occurrences.

Here is the complete list by number of occurrences from the alerts file.

```
109258 SMB Name Wildcard
 39363 Watchlist 000220 IL-ISDNNET-990517
 18176 High port 65535 udp - possible Red Worm - traffic
 14165 spp_http_decode: IIS Unicode attack detected
  9399 CS WEBSERVER - external web traffic
  8453 High port 65535 tcp - possible Red Worm - traffic
  7591 Tiny Fragments - Possible Hostile Activity
  5564 Incomplete Packet Fragments Discarded
  3965 TFTP - Internal TCP connection to external tftp server
  3951 TCP SRC and DST outside network
  3086 xxx.yyy.30.4 activity
  2803 External RPC call
  2711 spp_http_decode: CGI Null Byte attack detected
  1610 Null scan!
  1254 Watchlist 000222 NET-NCFC
  1175 Queso fingerprint
   854 IDS552/web-iis_IIS ISAPI Overflow ida nosize
   687 SUNRPC highport access!
   548 Possible trojan server activity
   473 EXPLOIT x86 NOOP
   307 CS WEBSERVER - external ftp traffic
   304 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize
   220 xxx.yyy.30.3 activity
   214 NMAP TCP ping!
   181 connect to 515 from outside
   170 DDOS mstream handler to client
   128 SNMP public access
   117 EXPLOIT x86 setuid 0
   114 NIMDA - Attempt to execute cmd from campus host
    85 IRC evil - running XDCC
    79 EXPLOIT x86 setgid 0
    75 TFTP - Internal UDP connection to external tftp server
    74 EXPLOIT x86 stealth noop
    71 DDOS mstream client to handler
    26 TFTP - External TCP connection to internal tftp server
    22 Notify Brian B. 3.56 tcp
    22 Notify Brian B. 3.54 tcp
    16 Attempted Sun RPC high port access
    11 SMB C access
     8 NETBIOS NT NULL session
     7 TCP SMTP Source Port traffic
     7 FTP passwd attempt
     5 Probable NMAP fingerprint attempt
     3 EXPLOIT NTPDX buffer overflow
     2 SYN-FIN scan!
     2 RFB - Possible WinVNC - 010708-1
     2 NIMDA - Attempt to execute root from campus host
     2 EXPLOIT digital unix noop
     1 Trin00 password on tcp
     1 External FTP to HelpDesk xxx.yyy.53.29
     1 DDOS shaft client to handler
     1 Bugbear@MM virus in SMTP
     1 Back Orifice
```

4.1 SMB Name Wildcard

This can be normal traffic generated by Windows computers to collect information about other devices on the network. It can also be used to perform reconnaissance to collect intelligence to launch an attack at a later date. The important distinction is if the traffic is being generated from the inside of the network or the outside. Because this is a possible information leak this traffic should not be allowed to enter or exit the network. External attempts should be viewed as hostile activity since it is an attempt to reconn the network or attack a system. Specifically, the 911 worm executes the "net view //" which generates a SMB wildcard packet. Out of the 109,258 packets 22,976 unique sources and 35,752 unique destinations. All the packets were from the external IP addresses entering the network.

Recommendation: Implement filter on the border router to eliminate this traffic from entering the network.

References: http://archives.neohapsis.com/archives/snort/2000-01/0222.html
http://www.cert.org/incident_notes/IN-2000-03.html

4.2 Watchlist 000220 IL-ISDNNET-990517

These 39,363 packets appear to be alerted on because of their source IP network address 212.179.0.0. As the name suggests in the alarm description, these packets are coming from ISDNNET. ISDNNET is registered to a company in Israel.

```
BEZEQINT HOSTMASTERS TEAM
bezeq-international
40 hashacham
petach tikva, 49170, Israel
```

A search for other alerts triggered by 212.179 showed 142 other alerts from this network. Most of the alerts were from SMB Wildcard alerts, but there was a NMAP scan, 3 possible red worm, and a CS WEBSERVER - external web traffic alert. A search for the destination address with 212.79 in the noScansAlerts file showed no packets. A search in the scans file showed the following.

```
Apr 5 09:37:57 xxx.yyy.194.223:55909 -> 212.179.201.233:5062 SYN ******S*
Apr 5 09:52:45 xxx.yyy.194.223:60754 -> 212.179.201.233:5062 SYN ******S*
Apr 5 09:44:52 xxx.yyy.194.223:57996 -> 212.179.201.233:5062 SYN ******S*
Apr 5 15:28:52 xxx.yyy.194.247:13139 -> 212.179.220.217:13139 UDP
```

This is cause for concern since the xxx.yyy of my.net is originating these packets.

I could not find any listing for TCP port 5062. James Bliss found that the UDP 13139 was from someone installing gamespy on their computer.

Http://lists.suse.com/archive/suse-security/2002-Feb/0374.html

The Internet Storm Center lists this port as registered for neverwinternights gaming server.

http://isc.incidents.org/port_details.html?port=13139

Recommendations: The 212.179 network must have a history of scanning the network. Sending an e-mail abuse@bezeqint.net complaining about the large amount of scans being performed. If no access to the UNIVERSITY is required then block the address range. The two devices, xxx.yyy.194.223 and xxx.yyy.194.247 that are initiating sessions need to be reviewed.

4.3 High port 65535 udp - possible Red Worm - traffic

The alerts file contained 18,176 of these alerts, with 214 unique sources to 248 unique destinations. Out of the 18,176 alerts, 7,883 were from the internal device xxx.yyy.201.58. I could not find the signature in the alert rules file. Sixty-six other internal systems alerted this rule. The following internal address were also involved in other alerts,

xxx.yyy.222.110, xxx.yyy.91.108, xxx.yyy.210.130, xxx.yyy.242.42, xxx.yyy.210.222, xxx.yyy.87.70, xxx.yyy.233.78, xxx.yyy.206.74, xxx.yyy.163.135.

The three internal systems xxx.yyy.242.42, xxx.yyy.210.222 and xxx.yyy.87.70 also appeared as accessing an external TFTP server with the address 24.236.251.22 and 209.126.214.14 which resolve to

OrgName: Charter Communications, Michigan Region
OrgID: CC03
Address: 359 US Hwy 41 East
City: Negaunee
StateProv: MI
PostalCode: 49866
Country: US

And

California Regional Internet, Inc. CARI

The Red Worm affects Linux systems. The port 65535 is the port the worm opens when it is activated. It also sends out e-mail to four different e-mail addresses.

Recommendations: Block pings from the outside to prevent the worm from going active. Monitor port 25 traffic for e-mail being sent from a system that shouldn't be sending out e-mail. Investigate all the 9 systems listed in this alert since it appears the UNIVERSITY network maybe is infected with the Red worm virus.

References: http://www.sans.org/rr/malicious/code_red8.php
http://www.sans.org/rr/casestudies/outbound.php
http://www.giac.org/practical/GCIA/Susan_Kovacevich_GCIA.pdf
http://www.europe.f-secure.com/v-descs/adore.shtml

4.4 spp_http_decode: IIS Unicode attack detected

The alerts file contained 14,165 of these alerts with 765 unique sources to 904 unique destinations. Out of these 14,165 alerts, 13,551 were from the internal network from 528 unique internal devices. Of these 528 internal sources 46 triggered multiply alerts. The 46 addresses are as follows

xxx.yyy.97.48, xxx.yyy.153.119, xxx.yyy.226.206, xxx.yyy.97.88, xxx.yyy.206.74, xxx.yyy.97.86, xxx.yyy.206.14, xxx.yyy.242.14, xxx.yyy.84.235, xxx.yyy.189.37, xxx.yyy.153.112, xxx.yyy.104.57, xxx.yyy.201.234, xxx.yyy.152.11, xxx.yyy.240.70, xxx.yyy.97.126, xxx.yyy.97.68, xxx.yyy.98.11, xxx.yyy.97.96, xxx.yyy.236.10, xxx.yyy.108.34, xxx.yyy.222.110, xxx.yyy.84.147, xxx.yyy.88.193, xxx.yyy.233.78, xxx.yyy.97.45, xxx.yyy.86.110, xxx.yyy.210.130, xxx.yyy.250.254, xxx.yyy.88.182, xxx.yyy.97.169, xxx.yyy.163.135, xxx.yyy.196.161, xxx.yyy.19.11, xxx.yyy.129.212, xxx.yyy.152.101, xxx.yyy.203.234, xxx.yyy.97.21, xxx.yyy.97.14, xxx.yyy.217.30, xxx.yyy.152.177, xxx.yyy.220.110, xxx.yyy.86.102, xxx.yyy.189.45, xxx.yyy.97.101, xxx.yyy.87.70, xxx.yyy.240.14

The following internal addresses also alerted for connecting to external TFTP servers as follow,

xxx.yyy189.37 to 80.212.98.5,
xxx.yyy.236.10 to 205.188.11.236, 205.188.6.52 and 64.12.26.248,
xxx.yyy.84.147 to 4.60.136.43,
xxx.yyy.189.45 to 4.60.136.43,
xxx.yyy.87.70 to 259.126.214.14.

The following two internal addresses also alerted for IRC evil connections, xxx.yyy.203.234 and xxx.yyy.220.110.

The following system was target by an external system according to the oos file may have been successfully compromised xxx.yyy.153.112

The http IIS Unicode attack works by passing file representation characters '../' or '..\' in Unicode. The Microsoft IIS security server performs a security check to make sure the request doesn't use any '../' to go outside the normal inetpub directory, but doesn't check to see if the unicode contains any of these characters. This is fine but the Snort preprocessor doesn't know what is outside the inetpub so this maybe normal traffic.

The following internal devices were also recorded in participating in scanning activity and should be investigated.

```
spp_http_decode: IIS Unicode attack detected
xxx.yyy.104.117  xxx.yyy.104.118   xxx.yyy.104.119   xxx.yyy.104.120
xxx.yyy.104.121  xxx.yyy.104.124   xxx.yyy.104.126   xxx.yyy.108.34
xxx.yyy.143.107  xxx.yyy.144.51    xxx.yyy.153.10    xxx.yyy.153.105
xxx.yyy.153.106  xxx.yyy.153.107   xxx.yyy.153.108   xxx.yyy.153.109
xxx.yyy.153.110  xxx.yyy.153.111   xxx.yyy.153.112   xxx.yyy.153.114
xxx.yyy.153.115  xxx.yyy.153.117   xxx.yyy.153.118   xxx.yyy.153.119
xxx.yyy.153.120  xxx.yyy.153.121   xxx.yyy.153.122   xxx.yyy.153.123
xxx.yyy.153.124  xxx.yyy.153.125   xxx.yyy.153.126   xxx.yyy.153.127
xxx.yyy.153.136  xxx.yyy.153.145   xxx.yyy.153.150   xxx.yyy.153.159
xxx.yyy.153.165  xxx.yyy.153.170   xxx.yyy.153.176   xxx.yyy.153.177
```

xxx.yyy.153.180	xxx.yyy.153.182	xxx.yyy.153.185	xxx.yyy.153.186
xxx.yyy.153.187	xxx.yyy.153.188	xxx.yyy.153.198	xxx.yyy.153.199
xxx.yyy.153.201	xxx.yyy.153.203	xxx.yyy.153.206	xxx.yyy.153.209
xxx.yyy.153.213	xxx.yyy.153.45	xxx.yyy.153.46	xxx.yyy.153.71
xxx.yyy.163.125	xxx.yyy.163.135	xxx.yyy.163.78	xxx.yyy.168.179
xxx.yyy.168.28	xxx.yyy.168.29	xxx.yyy.168.70	xxx.yyy.168.75
xxx.yyy.183.25	xxx.yyy.183.59	xxx.yyy.189.37	xxx.yyy.189.45
xxx.yyy.193.211	xxx.yyy.193.217	xxx.yyy.193.53	xxx.yyy.194.189
xxx.yyy.194.191	xxx.yyy.194.227	xxx.yyy.194.5	xxx.yyy.196.161
xxx.yyy.197.22	xxx.yyy.209.154	xxx.yyy.210.254	xxx.yyy.212.82
xxx.yyy.221.78	xxx.yyy.222.110	xxx.yyy.224.114	xxx.yyy.233.78
xxx.yyy.236.50	xxx.yyy.236.90	xxx.yyy.240.38	xxx.yyy.240.70
xxx.yyy.242.14	xxx.yyy.242.250	xxx.yyy.54.210	xxx.yyy.84.147
xxx.yyy.86.110	xxx.yyy.87.70	xxx.yyy.88.156	xxx.yyy.88.229
xxx.yyy.97.102	xxx.yyy.97.105	xxx.yyy.97.13	xxx.yyy.97.133
xxx.yyy.97.134	xxx.yyy.97.136	xxx.yyy.97.138	xxx.yyy.97.139
xxx.yyy.97.140	xxx.yyy.97.149	xxx.yyy.97.155	xxx.yyy.97.165
xxx.yyy.97.170	xxx.yyy.97.171	xxx.yyy.97.177	xxx.yyy.97.178
xxx.yyy.97.188	xxx.yyy.97.198	xxx.yyy.97.202	xxx.yyy.97.203
xxx.yyy.97.226	xxx.yyy.97.230	xxx.yyy.97.233	xxx.yyy.97.246
xxx.yyy.97.31	xxx.yyy.97.36	xxx.yyy.97.37	xxx.yyy.97.62
xxx.yyy.97.75	xxx.yyy.97.86	xxx.yyy.97.88	xxx.yyy.97.92
xxx.yyy.97.98	xxx.yyy.98.16		

Recommendations: Check the systems listed above since they had multiply alerts. Determine if the TFTP connections listed above were authorized. Correlate the packets to only the servers listening open on the firewall for http within the network and log the remainder since they will not succeed.

References: http://www.snort.org/docs/writing_rules/chap2.html#tth_sEc2.4.1
http://www.snort.org/docs/faq.html#4.17
http://lists.suse.com/archive/suse-security/2001-Mar/0371.html

4.5 CS WEBSERVER - external web traffic

The alerts file contained 9,399 of these alerts with 4,261 unique sources to 1 unique destination. The 1 unique destination is xxx.yyy.100.65, which is resolved to www.UNIVERSITY.edu/engineer/cse. I could not find this rule in the default ruleset. This appears to be a rule to capture packets that are not from the xxx.yyy.0.0 destined for the www.UNIVERSITY.edu/engineer/cse as the name implies.

Recommendations: Change the rule from alert to log to avoid filling up the alert log but not loosing the logs of who is accessing the www.UNIVERSITY.edu/engineer/cse web server.

References: www.UNIVERSITY.edu/engineer/cse

4.6 High port 65535 tcp - possible Red Worm - traffic

The alerts file contained 8,453 of these alerts, with 123 unique sources to 119 unique destinations. Out of the 8,453 alerts, 53 were from the internal device. The following 17 internal address were also involved in other alerts. The addresses are as follows.

xxx.yyy.86.110, xxx.yyy.226.206, xxx.yyy.201.234, xxx.yyy.202.206, xxx.yyy.240.14, xxx.yyy.87.70, xxx.yyy.105.48, xxx.yyy.100.230, xxx.yyy.249.134, xxx.yyy.88.193, xxx.yyy.194.13, xxx.yyy.24.33, xxx.yyy.202.222, xxx.yyy.233.78, xxx.yyy.242.14, xxx.yyy.86.102, xxx.yyy.189.37.

The internal systems xxx.yyy.105.48 appears to be acting as an TFTP server, and xxx.yyy.86.102 appears to be as accessing an external TFTP server with the address 80.212.98.5 which is registered as

netname: NO-NEXTRA-ADSL-1
descr: Telenor Business Solution AS
country: NO

The 17 internal addresses are systems that have been infected and are now infection other machines.

The following internal devices were also recorded in participating in scanning activity and should be investigated.

xxx.yyy.100.230 xxx.yyy.189.37 xxx.yyy.194.13 xxx.yyy.194.223 xxx.yyy.224.90 xxx.yyy.233.78
xxx.yyy.234.174 xxx.yyy.242.14 xxx.yyy.250.226 xxx.yyy.86.110 xxx.yyy.87.70 xxx.yyy.163.135
xxx.yyy.193.213 xxx.yyy.195.209 xxx.yyy.201.218 xxx.yyy.203.46 xxx.yyy.204.8 xxx.yyy.205.198
xxx.yyy.208.66 xxx.yyy.210.222 xxx.yyy.217.178 xxx.yyy.222.110 xxx.yyy.224.170 xxx.yyy.228.50
xxx.yyy.233.78 xxx.yyy.237.226 xxx.yyy.238.214 xxx.yyy.241.78 xxx.yyy.242.42 xxx.yyy.251.126
xxx.yyy.251.30 xxx.yyy.253.102 xxx.yyy.87.70 xxx.yyy.91.109

The Red Worm affects Linux systems. The port 65535 is the port the worm opens when it is

Recommendations: Block pings from the outside to prevent the worm from going active. Monitor port 25 traffic for e-mails being sent from a system that shouldn't be sending out e-mails. Investigate all the 17 systems listed in this alert since it appears the UNIVERSITY network is infected with the Red worm virus.

References: http://www.sans.org/rr/malicious/code_red8.php
http://www.sans.org/rr/casestudies/outbound.php
http://www.giac.org/practical/GCIA/Susan_Kovacevich_GCIA.pdf
http://www.europe.f-secure.com/v-descs/adore.shtml

4.7 Tiny Fragments - Possible Hostile Activity

The alerts file contained 7,591 of these alerts, with 8 unique sources to 224 unique destinations. The rule that triggered it appears below but it should only alert when a packet from the external network is destined for the xxx.yyy.0.0 network, although 7,480 alerts were logged with xxx.yyy.240.78 as the source. This is probably an indication that the $HOME_NET variable is not defined or the rule was modified. The default rule is listed below.

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC Tiny Fragments";
fragbits:M; dsize: < 25; classtype:bad-unknown; sid:522; rev:1;)
```

An E-mail from Jeff Oxenreider describes his finding GNUTELLA as generating these types of packets.

The following internal devices were also recorded in participating in scanning activity and should be investigated.

xxx.yyy.240.78 xxx.yyy.240.78

Recommendations: Capture complete packets to investigate the contents.

References: http://www.snort.org/snort-db/sid.html?sid=522
http://archives.neohapsis.com/archives/snort/2000-05/0115.html

4.8 Incomplete Packet Fragments Discarded

The alerts file contained 5,564 of these alerts with 67 unique sources to 53 unique destinations. I could not find the rule in the default snort rules. The packets all appear to have a source port of 0 and a destination port of 0. Both TCP and UDP port 0 is listed on the IANA website as reserved. Of the 5,564 alerts, 5,055 were from one address xxx.yyy.252.166 to the address 63.210.47.23. This activity occurred within a two-hour period from 09:09:43 on 04/05/2003 to 11:16:16 on 04/05/2003. Other source and destination also seem to occur in short bursts but only between devices. The use of port 0, the short burst of traffic and specific targets indicates a denial of service attack. A look at some of the captured packets may reveal this in not the case.

The following internal device was also recorded in participating in scanning activity and should be investigated, xxx.yyy.252.166

Recommendations: Capture some of these packets to determine if it is an actual attack or just a broken program. Since port 0 is reserved, it should not be on the network. Block all TCP and UDP packets with a source or destination port of 0. Review the configuration and programs installed on xxx.yyy.252.166.

Contact the owner of 63.210.47.23 to see if they have any logs of this activity. It is listening on port 80 and responds with "unknown.Level3.net". It is registered to Level 3 Communications, 1025 Eldorado Blvd., Broomfield, CO, 80021, US. Complaints should be sent to abuse@level3.com

References: http://www.iana.org/assignments/port-numbers

4.9 TFTP - Internal TCP connection to external tftp server

The alerts file contained 3,965 of these alerts with 36 unique sources to 34 unique destinations. I could not find the rule in the default snort rules. The packets all appears to have a source port of 69 and a destination IP network of xxx.yyy.0.0 or a destination port of

69 and a source IP network of xxx.yyy.0.0. This alerts when a packet is sent to a TFTP server from the internal network or when a packet from a TFTP server is sent to the internal network.

A test of two of the external addresses showed them as TFTP servers. These appear to be valid TFTP sessions from the internal network to external sites. TFTP file transfers can allow the download of malicious code. Many vulnerabilities are associated with TFTP. A listing at http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=TFTP provide further information on the types. TFTP is also used by NIMDA to download the worm to the IIS server.

The following internal devices were also recorded in participating in scanning activity and should be investigated.

```
TFTP - Internal TCP connection to external tftp server
xxx.yyy.189.37   xxx.yyy.132.23    xxx.yyy.178.19    xxx.yyy.189.45
xxx.yyy.210.222  xxx.yyy.236.90    xxx.yyy.242.42    xxx.yyy.84.147
xxx.yyy.87.70    xxx.yyy.91.109
```

Recommendation: Block access from the internal network to TFTP servers or capture the files beginning downloaded to check for malicious content. Ascertain if the TFTP servers are used for authorized or malicious purposes.

References: http://www.portsdb.org/bin/portsdb.cgi?portnumber=69&protocol=ANY
http://www.cert.org/advisories/CA-2001-26.html

4.10 TCP SRC and DST outside network

The alerts file contained 3,951 of these alerts with 1,928 unique sources to 81 unique destinations. I could not find the rule in the default snort rules. It appears that the packets do not contain the internal network address of xxx.yyy.0.0 as the source or destination. I suspect these are packets from within the internal network that are using private address, misconfigured devices or spoofed address.

Out of the almost 4,000 alerts, 235 contain the private address of 192.168.0.0, 7 contain the private address 172.28.0.0, 603 contain the private address 252.0.0.0 and 1,269 contain the private address of 251.255.0.0.

Misconfigured packets would include 42 alerts with the address 0.0.0.0

Some 1,745 alerts were from the source address network of 207.46.0.0. This address is registered to Microsoft. These where probably spoofed by someone from the internal network.

Recommendations: Institute anti-spoofing on the network security device to disallow any packets from leaving the network if they don't have an internal source address.

References:http://ws.arin.net/cgi-bin/whois.pl

4.11 xxx.yyy.30.4 activity

The alerts file contained 3,086 of these alerts with 257 unique sources to 1 unique destination. I could not find the rule in the default snort rules. As name indicates, the alert appears to trigger for every packet destined for the internal device xxx.yyy.30.4. A look at the log file showed most of the packets has a destination port 80. A quick check with my favorite browser showed this site as the UNIVERSITY Novell Netstorage device.

Recommendations: Log these packets rather than alert.

References: http://xxx.yyy.30.4/

4.12 External RPC call

The alerts file contained 2803 of these alerts with 16 unique sources to 2,747 unique destinations. I could not find the rule in the default snort rules. The name of the alert and packets captured with it implies the rule alerts on packets from $EXTERNAL_NET to $INTERNAL_NET with a destination port of 111. Out of the 2,802 alerts, 2,788 were received from the two sources **62.65.192.30** (1811) and **203.197.255.90** (977). The many different address is an indication these systems were performing a scan for a server listening for RPC portmapper requests. Portmapper is used by RPC to identify which services are running on what high number ports. This information can be used to attempt to connect to the RPC services. The first scan was performed in a little over three minutes the second was done on two different days in less than a minute for the first and less than three minutes for the second scan. Neither of the addresses appeared in the scans file as scanning any other systems nor where they involved in any other alerts. Concern would be in order if these two addressees attempted access to the commonly used RPC high ports 32771 – 32789.

Recommendations: Block destination port 111 from entering the network.

4.13 spp_http_decode: CGI Null Byte attack detected

The alerts file contained 2,711 of these alerts with 127 unique sources to 119 unique destinations. The alert is generated by the snort preprocessor for http decodes. This alert is an indication that a “%00” was detected in the content. This can be a false positive from cookies with urlencoded binary.

Of the 127 unique sources 37 internal systems are involved in other alerts. The majority of other alerts are the spp_http_decode: IIS Unicode attack detected. The following lists the systems that had other alerts.

xxx.yyy.97.48	xxx.yyy.97.99	xxx.yyy.98.11	xxx.yyy.153.112	xxx.yyy.97.96	xxx.yyy.97.21
xxx.yyy.97.68	xxx.yyy.97.14	xxx.yyy.217.30	xxx.yyy.202.206	xxx.yyy.84.185	xxx.yyy.218.214
xxx.yyy.152.177	xxx.yyy.108.34	xxx.yyy.153.119	xxx.yyy.226.206	xxx.yyy.97.86	xxx.yyy.97.126
xxx.yyy.229.18	xxx.yyy.233.78	xxx.yyy.206.14	xxx.yyy.152.11	xxx.yyy.236.90	xxx.yyy.153.122
xxx.yyy.97.45	xxx.yyy.97.169	xxx.yyy.199.250	xxx.yyy.86.102	xxx.yyy.97.101	xxx.yyy.183.59
xxx.yyy.201.234	xxx.yyy.84.235	xxx.yyy.91.109	xxx.yyy.98.23	xxx.yyy.88.182	xxx.yyy.163.135
xxx.yyy.250.254					

Out of these 37, 6 internal addresses also alerted on possible Red worm. The addresses are

xxx.yyy.202.206 xxx.yyy226.206 xxx.yyy.233.78 xxx.yyy.86.102 xxx.yyy201.234 xxx.yyy163.135

Out o the same 37 above, 2 internal addresses also alerted on TFTP to external site. The internal addresses are

```
xxx.yyy.236.90 xxx.yyy.91.109
```

Both addresses accessed the same external server 209.126.214.14. Several other internal addresses access this TFTP server as well as listed below.

```
TFTP - Internal UDP connection to external tftp server [**]
xxx.yyy.210.222:11544 -> 209.126.214.14:69
TFTP - Internal UDP connection to external tftp server [**]
xxx.yyy.210.222:11544 -> 209.126.214.14:69
TFTP - Internal UDP connection to external tftp server [**] xxx.yyy.87.70:9634
-> 209.126.214.14:69
```

The following system was targeted according to the oos file and may have been successfully compromised since appears it is attacking other systems. xxx.yyy.97.45.

The address 209.126.214.14 is registered to California Regional Internet, Inc. CARI.

The following internal devices were also recorded in participating in scanning activity and should be investigated.

xxx.yyy.108.34 xxx.yyy.114.252 xxx.yyy.143.154 xxx.yyy.153.112 xxx.yyy.153.119 xxx.yyy.153.122
xxx.yyy.163.135 xxx.yyy.183.59 xxx.yyy.196.161 xxx.yyy.205.174 xxx.yyy.233.78 xxx.yyy.236.62
xxx.yyy.236.90 xxx.yyy.91.109 xxx.yyy.97.132 xxx.yyy.97.173 xxx.yyy.97.38 xxx.yyy.97.86
xxx.yyy.98.169

Recommendations: Capture the content to determine if this is an attack or false positive. Limit captures only to internal HTTP servers to reduce false positives. Determine if the 209.126.214.14 is being used by hackers or authorized purposes.

References: http://archives.neohapsis.com/archives/snort/2000-11/0244.html

4.14 Null scan!

The alerts file contained 1,160 of these alerts with 68 unique sources to 78 unique destinations. All the packets are coming from the external to the internal network. It appeared at first that all the packets had a source and destination port of 0. After looking at all the packets I found 212.202.193.59:1658 -> xxx.yyy.237.6:1872. I looked for this packet in the scan file and found 212.202.193.59:1658 -> xxx.yyy.237.6:1872 NULL ********. along with the other null scan packets. This snort rule alerts when no TCP flags are set.

Recommendations: Block all packets entering the network with no TCP flags.

4.15 Watchlist 000222 NET-NCFC

The alerts file contained 1,254 of these alerts. The packets appear to be alerted on because of their source IP network address 159.226.0.0. I could not find the rule in the default snort

rules. These packets are coming from Computer Network Center Chinese Academy of Sciences. ISDNNET is registered to a company in China.

```
The Computer Network Center Chinese Academy of Sciences
Address: P.O. Box 2704-10
Institute of Computing Technology Chinese Academy of Sciences
Beijing 100080, China
```

A search for other alerts triggered by a 159.226.0.0 showed 17 other alerts from this network. Most of the alerts were from SMB Wildcard, but there was a Queso, and a CS WEBSERVER - external web traffic alert. A search for address with 159.226.0.0 in the oos and scans gave me 14 packets.

```
oos130:04/01-22:02:20.111923 159.226.113.140:44755 -> xxx.yyy.130.14:80
oos130:04/01-22:47:18.817356 159.226.113.140:46066 -> xxx.yyy.130.14:80
oos130:04/02-02:35:54.042964 159.226.113.140:53650 -> xxx.yyy.130.14:80
oos130:04/02-02:36:02.562933 159.226.113.140:53651 -> xxx.yyy.130.14:80
oos130:04/02-04:33:49.026361 159.226.113.140:57370 -> xxx.yyy.130.14:80
oos130:04/02-07:12:29.094078 159.226.162.168:1357 -> xxx.yyy.100.237:9080
scans_130:Apr  2 02:36:02 159.226.113.140:53651 -> xxx.yyy.130.14:80 SYN
12****S* RESERVEDBITS
scans_130:Apr  2 05:16:17 xxx.yyy.97.88:3915 -> 159.226.155.101:80 SYN ******S*
scans_130:Apr  4 04:03:06 xxx.yyy.1.3:32790 -> 159.226.1.1:53 UDP
scans_130:Apr  4 04:03:11 xxx.yyy.1.3:32790 -> 159.226.118.1:53 UDP
scans_130:Apr  4 04:03:21 xxx.yyy.1.3:32790 -> 159.226.1.1:53 UDP
scans_130:Apr  4 04:03:22 xxx.yyy.1.3:32790 -> 159.226.1.1:53 UDP
scans_130:Apr  4 04:03:25 xxx.yyy.1.3:32790 -> 159.226.1.1:53 UDP
scans_130:Apr  5 02:55:51 xxx.yyy.137.7:29460 -> 159.226.1.1:53 UDP
```

Recommendations: The 159.226.0.0 network has a history of scanning the network. If no access to the UNIVERSITY is required then block the address range. No contact information was listed at the ARIN website.

References: http://ws.arin.net/cgi-bin/whois.pl

4.16 Queso fingerprint

The alerts file contained 1,175 of these alerts with 293 unique sources to 110 unique destinations. I could not find the rule in the default snort rules. A look at the Queso fingerprint alerts for the file didn't reveal anything unusual but the search for one of the alerts source/destination pair showed the following.

```
Apr  1 07:20:03 193.232.119.109:47513 -> xxx.yyy.221.194:6881 SYN 12****S*
RESERVEDBITS
```

A search for all 12****S* strings in the scans file showed the same number as Queso alerts. It appears this snort rule triggers when the first two reserved bits are set along with the Syn bit. These are probably false positives due to diverse number of source addresses and the new implementation of rfc3168 to detect congestion. This was also recorded in the oos file

Recommendation: Tune the IDS to ignore these packets and filter the illegal combinations at the firewall. Even without the ECN bits the flag combination is illegal since it doesn't contain

an ACK or SYN flag. A more detailed description is give for these flag combinations in the oos analysis section.

Reference:http://www.icir.org/floyd/papers/rfc3168.txt

4.17 IDS552/web-iis_IIS ISAPI Overflow ida nosize

The alerts file contained 854 of these alerts with 607 unique sources to unique 699 destinations. Most of the packets are from external addresses with the exception of three. The snort rule that probably triggered is below:

```
alert TCP $EXTERNAL any -> $INTERNAL 80 (msg: "IDS552/web-iis_IIS ISAPI
Overflow ida"; dsize: >239; flags: A+; uricontent: ".ida?"; classtype:
system-or-info-attempt; reference: arachnids,552;)
```

This event shows someone maybe trying to exploit a flaw in Microsoft IIS. An index server on the Microsoft IIS Index Server has a buffer that doesn't check for size. An attacker can gain system access through this exploit. This may also be a false positive since it alerts any time a packet that contains ".ida?", ACK flag and others, and data over 239.

Recommendations: Ensure web servers are patched. Block port 80 access to any device except know hardened web servers that require external access.

References: http://www.giac.org/practical/Edward_Peck_GCIA.doc
http://www.whitehats.com/IDS/552

4.18 SUNRPC highport access!

The alerts file contained 687 of these alerts with 65 unique sources to 42 unique destinations. Most of the packets are from the external addresses with the exception of one. The snort rule that probably trigger is below:

```
alert tcp any any -> any 32771 (msg: "SUNRPC highport access!";)
```

The rule is triggered every time a packet has a destination port number 32771. Since this is an ephemeral port number it can be used in valid traffic. The pattern of one to one and short time span between the packets indicates a scan for an open port was not being performed.

Recommendation: Modify rule to only alert on packets destined for the internal network. Block access to all Sun servers that don't require external access.

References: http://h30097.www3.hp.com/demos/ossc/doc/snort-1.8p1/RULES.SAMPLE

4.19 Possible trojan server activity

The alerts file contained 548 of these alerts with 72 unique sources to 47 unique destinations. Packets are from the both the external and internal addresses. The rule is triggered when a packet has a source or destination port number 27374. Since this is an ephemeral port number it can be used in valid traffic. This is a well-known TCP port for SubSeven 2.1.

The Internal address the with the number of each occurrences is as follows

```
190 xxx.yyy.240.70      131 xxx.yyy.226.226     67 xxx.yyy.223.226
11 xxx.yyy.6.7          2 xxx.yyy.24.33         2 xxx.yyy.24.20
2 xxx.yyy.222.110       2 xxx.yyy.201.106       2 xxx.yyy.194.13
1 xxx.yyy.249.134       1 xxx.yyy.24.341        1 xxx.yyy.194.117
1 xxx.yyy.150.133       1 xxx.yyy.100.230
```

The following internal devices were also recorded in participating in scanning activity and should be investigated.

Possible trojan server activity

```
xxx.yyy.100.230   xxx.yyy.194.117   xxx.yyy.194.13    xxx.yyy.201.106
 xxx.yyy.222.110  xxx.yyy.240.70
```

Recommendation: Modify rule to only alert on a packet with a source of 27374 from the internal network. Ensure all windows platforms are running current patches and anti-virus software. Inspect all internal systems to ensure they are not harboring a trojan program.

References: http://www.sans.org/y2k/subseven.htm

4.20 EXPLOIT x86 NOOP

The alerts file contained 473 of these alerts with 116 unique sources to 95 unique destinations. All the packets are from external addresses. The snort rule that probably trigger is below:

```
alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE x86
inc ebx NOOP"; content:"|43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 43 43|"; classtype:shellcode-detect; sid:1390; rev:3;)
```

The alert is triggered from the content of a packet. Like any content filter it is suspect to false positives.

Recommendation: Block all access to internal web servers that don't require external access. Ensure the web servers are patched and hardened.

Reference: http://archives.neohapsis.com/archives/sf/ids/2002-q2/0018.html

4.21 CS WEBSERVER - external ftp traffic

The alerts file contained 307 of these alerts with 115 unique sources to 1 unique destination. All the packets are from external addresses. It appears the rule triggers on packets with a destination port of 21.

Recommendations: Block all access to internal FTP servers that don't require external access. Ensure public FTP servers are hardened.

4.22 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize

The alerts file contained 304 of these alerts with 4 unique sources to 264 unique destinations. All the packets are from external addresses. The reference for this is the same as 4.17 854 IDS552/web-iis_IIS ISAPI Overflow ida nosize.

The following internal devices were also recorded in participating in scanning activity and should be investigated.

```
xxx.yyy.97.66   xxx.yyy.97.88     xxx.yyy.98.157     xxx.yyy.98.184
```

Recommendations: Ensure web servers are patched. Block port 80 access to any device except know hardened web servers that require external access.

Reference: http://www.whitehats.com/IDS/552

4.23 xxx.yyy.30.3 activity

The alerts file contained 220 of these alerts with 44 unique sources to 1 unique destination. The one destination address is xxx.yyy.30.3 as the name of the alert implies. All the packets are from external addresses. Destination ports are 80, 119, 139, 445, 524, 1433, and 3128. Most of the ports are 524. Port 524 is associated with NetWare file services. The xxx.yyy.30.3 is open to the public and seems to have the default installation page. It appears similar to the xxx.yyy.30.4 activity.

Recommendations: Ensure the web server is patched and hardened. Replace default web page to hide the fact it is a NetWare server.

References: http://www.geocrawler.com/archives/3/90/2001/9/0/6722577/

4.24 NMAP TCP ping!

The alerts file contained 214 of these alerts with 45 unique sources to 76 unique destinations. The time interval and one to one correspondence shows this was not a scan. I could not locate the rule in the default snort rules. I found a rule on the web as shown below.

```
tcp any -> any msg:"IDS028 - PING NMAP TCP"; flags:A; ack:0; dlevel: 1;
```

It alerts when a packet has the ack bit set and the ack number of 0.

Recommendations: Watch the traffic pattern to determine if a scan is taking place.

Reference:
http://www.cipherdyne.com/cgi/viewcvs.cgi/psad/psad_signatures.diff?r1=1.19&r2=1.20

4.25 connect to 515 from outside

The alerts file contained 181 of these alerts with 24 unique sources to 3 unique destinations. All the packets are from external IP addresses destined for port 515. Port 515 is used for remote printing.

Recommendations: Block access to port 515 from external sources unless access is required. Identify what external addresses are required and restrict access to these addresses.

References: http://www.portsdb.org/bin/portsdb.cgi?portnumber=515&protocol=ANY

4.26 DDOS mstream handler to client

The alerts file contained 170 of these alerts with 1 unique source to 2 unique destinations. All the packets are from internal IP addresses destined for port 12754 or 15104. The two snort rules are below.

```
alert tcp $HOME_NET 12754 -> $EXTERNAL_NET any (msg:"DDOS mstream handler to
client"; content: ">"; flags: A+;reference:cve,CAN-2000-0138;
classtype:attempted-dos; sid:248; rev:1;)
alert tcp $HOME_NET 15104 -> $EXTERNAL_NET any (msg:"DDOS mstream handler to
client"; content: ">"; flags: A+; reference:cve,CAN-2000-0138;
classtype:attempted-dos; sid:250; rev:1;)
```

The alert triggers when packets from the internal network are destined for the external network with the port number 12754 or 15104 and the content of ">". The ">" is an indication of the system mstream program responding.

Recommendation: Capture the packets and search for the mserver commands, stream, servers, ping, who and mstream.

References:http://www.cipherdyne.com/cgi/viewcvs.cgi/psad/psad_signatures.diff?r1=1.6&r2=1.7

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-0138
http://www.ciac.org/ciac/bulletins/k-037.shtml

4.27 SNMP public access

The alerts file contained 128 of these alerts with 14 unique sources to 13 unique destinations. All the packets are from external IP addresses destined for port 161, which contain the string public. The snort rule is below.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 161 (msg:"SNMP public access udp";
content:"public"; reference:cve,CAN-1999-0517; reference:cve,CAN-2002-0012;
reference:cve,CAN-2002-0013; sid:1411; rev:3; classtype:attempted-recon;)
```

The public string is a common default string for SNMP read access.

Recommendations: Block all access into the network for SNMP access by blocking port 161. Verify that the 13 destination addresses do not contain the default read string of public and the write string of private.

References: http://www.portsdb.org/bin/portsdb.cgi?portnumber=161&protocol=ANY

4.28 EXPLOIT x86 setuid 0

The alerts file contained 117 of these alerts with 107 unique sources to 94 unique destinations. All the packets are from external IP addresses destined for internal IP addresses. It appears that the alert triggers when a packet with an external IP address is destined to an internal IP address with the content of setuid(0).

Recommendation: Capture the packet to analyze the content to determine if it is an attack or false positive.

References: http://www.securiteam.com/exploits/5JP0A1P9GQ.html

4.29 NIMDA - Attempt to execute cmd from campus host

The alerts file contained 114 of these alerts with 4 unique sources to 111 unique destinations. All the packets are from internal IP addresses destined for external IP addresses with destination port 80. The internal addresses are xxx.yyy.97.66, xxx.yyy.98.157, xxx.yyy.97.88 and xxx.yyy.98.184. I can't locate the signature in the default snort rules. All four sources triggered other alerts. The alert probably alerts on the string '/winnt/system32/cmd.exe?'. As any content alert these could be false positives but they also alerted on IIS ISAPI, IIS Unicode and Attempt to execute root from campus host.

The following internal devices were also recorded in participating in scanning activity and should be investigated.

```
xxx.yyy.97.66     xxx.yyy.97.88     xxx.yyy.98.157    xxx.yyy.98.184
```

Recommendation: Investigate all four systems for possible NIMDA infection. See section 4.46.

References: http://www.sans.org/rr/malicious/nimda3.php

4.30 IRC evil - running XDCC

The alerts file contained 85 of these alerts with 20 unique sources to 22 unique destinations. All the packets are from internal IP addresses destined for external IP addresses with destination port range 6665 – 6669 used by IRC. I could not locate the snort rule.

The following internal device was also recorded in participating in scanning activity and should be investigated, xxx.yyy.114.11 .

Recommendation: Block ports 6665 - 6669

References: http://www.russonline.net/tonikgin/EduHacking.html

4.31 EXPLOIT x86 setgid 0

The alerts file contained 79 of these alerts with 73 unique sources to 69 unique destinations. All the packets are from external IP addresses destined for internal IP addresses with various source and destination ports. It appears that the alert triggers when a packet with an external IP address is destined to an internal IP address with the content of setuid(0).

Recommendation: Capture the packet to analyze the content to determine if it is an attack or false positive.

4.32 TFTP - Internal UDP connection to external tftp server

The alerts file contained 75 of these alerts with 15 unique sources to 15 unique destinations. The packets are from both internal and external IP addresses, with port number 69 as the source or destination ports.

TFTP file transfers can allow the download of malicious code. Many vulnerabilities are associated with TFTP. A listing at http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=TFTP provide further information on the types. TFTP is also used by Nimda to download the worm to the IIS server.

Recommendation: Block access to internal TFTP servers unless required. Ensure TFTP servers are hardened.

References: http://www.cert.org/advisories/CA-2001-26.html

4.33 EXPLOIT x86 stealth noop

The alerts file contained 74 of these alerts with 16 unique sources to 15 unique destinations. All the packets are from external IP addresses destined for internal IP addresses with various source and destination ports. The alert is triggered from the packet content. One of the source addresses, 131.118.254.130, was the source for many of the same type of alerts. This address is registered to the University of Maryland.

Recommendations: Ensure all systems are patched. Capture packets to analysis the contents. Contact the University of Maryland to ensure they are not harboring a hacker.

4.34 DDOS mstream client to handler

The alerts file contained 71 of these alerts with 27 unique sources to 3 unique destinations. All the packets are from external IP addresses destined for internal IP address with port 12754 or 15104. The two snort rules are below.

```
alert tcp $EXTERNAL_NET  12754 -> $HOME_NET any (msg:"DDOS mstream client to
handler"; content: ">"; flags: A+;reference:cve,CAN-2000-0138;
classtype:attempted-dos; sid:247; rev:1;)

 alert tcp $EXTERNAL_NET 15104 -> $ HOME_NET  any (msg:"DDOS mstream client
 to handler"; content: ">"; flags: A+; reference:cve,CAN-2000-0138;
 classtype:attempted-dos; sid:249; rev:1;)
```

The alert triggers when packets from the external network are destined for the internal network with the port number 12754 or 15104 and the content of '>'. The ">" is an indication of the system mstream program responding.

Recommendation: Capture the packets and search for the mserver commands, stream, servers, ping, who and mstream.

References:http://www.cipherdyne.com/cgi/viewcvs.cgi/psad/psad_signatures.diff?r1=1.6&r2=1.7
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-0138
http://www.ciac.org/ciac/bulletins/k-037.shtml

4.35 TFTP - External TCP connection to internal tftp server

The alerts file contained 26 of these alerts with 13 unique sources to 15 unique destinations. The packets are from both internal and external IP addresses, with port number 69 as the source or destination port. The only internal address is xxx.yyy.105.48.

TFTP file transfers can allow the download of malicious code. Many vulnerabilities are associated with TFTP. A listing at http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=TFTP provides further information on the types. The Nimda worm also uses TFTP to transfer the worm to the IIS server.

Recommendation: Block access to internal TFTP servers, port 69, unless required. Ensure the public TFTP servers are authorized and hardened.

References: http://www.cert.org/advisories/CA-2001-26.html
cve.mitre.org/cgi-bin/cvekey.cgi?keyword=TFTP

4.36 Notify Brian B. 3.56 tcp and 22 Notify Brian B. 3.54 tcp

The alerts file contained 44 of these alerts, 22 for each. The 3.56 alerts contained 21 unique sources to 1 unique destination. The 3.56 alerts contained 19 unique sources to 1 unique destination. All the packets are from external IP addresses destined for two internal IP addresses xxx.yyy.3.56 and xxx.yyy.3.54 with any source or destination port. I suspect these are honeypots to catch unusual traffic and attacking IP address.

Recommendations: Watch log for other alerts from the same address. Ensure rules are in the correct order so these rules don't fire first.

4.37 Attempted Sun RPC high port access

The alerts file contained 19 of these alerts with 4 unique sources to 3 unique destinations. All the packets are from external IP addresses destined for internal IP addresses with destination port 32771. The port number is register as High Ports used in Solaris for RPC services.

Recommendations: Implement a stateful inspection firewall. Block external access not required.

4.38 SMB C access

The alerts file contained 11 of these alerts with 10 unique sources to 8 unique destinations. All the packets are from external IP addresses destined for internal IP addresses with destination port 139. The port number is register as NETBIOS-ssn.

Recommendations: Block all port 139 from entering the network.

4.39 NETBIOS NT NULL session

The alerts file contained 8 of these alerts with 1 unique source to 5 unique destinations. All the packets are from an external IP address destined for internal IP addresses with destination port 139.

The IP address range, 210.160.200.0 – 210.160.203.255, is registered by APNIC to a company in China as shown below.

```
HWT,
HanWang Technology Co.LTD,
Technology,
BeiJing, CN
```

Recommendation: Block the 210.160.200.0 – 210.160.203.255 range of addresses or closely monitor it. Investigate the xxx.yyy.30.3 to see if it is compromised.

4.40 TCP SMTP Source Port traffic

The alerts file contained 7 of these alerts with 1 unique source to 2 unique destinations. The packet is from an external IP address destined for internal IP addresses with source and destination port 25. Some attempts are made to use a source port of 25 to bypass a firewall and go to other ports. This is not the case since the destination port is 25 as well.

4.41 FTP passwd attempt

The alerts file contained 7 of these alerts with 1 unique source to 7 unique destinations. The packet is from external IP address destined for internal IP addresses with destination port 21.

The alert triggers when a packet is from the external to internal with a destination port of 21 and content of ' passwd'. Since this is a content alert, the packets should be captured for analyzed. The sources where not involved in other alerts.

4.42 Probable NMAP fingerprint attempt

The alerts file contained 5 of these alerts with 5 unique sources to 5 unique destinations. All the packets are from external IP addresses destined for internal IP addresses with various source and destination ports. Two of these source address, 212.159.60.110 and 61.145.139.65, are involved in other alerts. Both address are registered to the same place as shown below.

```
212.159.60.110
210.160.200.0 - 210.160.203.255
CHINANET-GD
CHINANET Guangdong province network
Data Communication Division
China Telecom
CN

61.145.139.65
61.145.0.0 - 61.145.255.255
HINANET-GD
CHINANET Guangdong province network
Data Communication Division
China Telecom
CN
```

Recommendations: Block all traffic from these networks.

References: http://www.giac.org/practical/Mike_Bell_GCIA.doc

4.43 EXPLOIT NTPDX buffer overflow

The alerts file contained 3 of these alerts with 3 unique sources to 3 unique destinations. All the packets are from external IP addresses destined for internal IP addresses with a source or destination port of 123.

Recommendations: Block all NTP port 123 accesses into the network except from a trusted source and to a hardened timeserver. All timeservers should reference this server or other internal servers.

Reference: http://www.xfocus.net/exploits/linux_ntpd.txt

4.44 SYN-FIN scan!

The alerts file 2 of these alerts with contained 1 unique source to 1 unique destination. The packet is from an external IP address destined for an internal IP addresses with both the syn and fin bits set. The source IP address **213.219.90.74** was involved in multiply other scans against the same address.

```
Estonian Telephone Co/Estpak Data Ltd.
Sole str 14, Tallinn, Estonia
```

Recommendations: Place a watch on this address. Block network address range 213.219.89.0 – 213.219.90.255.

4.45 RFB - Possible WinVNC - 010708-1

The alerts file contained 2 of these alerts with 2 unique sources to 2 unique destinations. The packets are from both internal IP addresses and external IP addresses with a source or destination port of 5900. WinVNC uses port 5900.

Recommendation: Scan internal IP addresses for systems listening on port 5900.

4.46 NIMDA - Attempt to execute root from campus host

The alerts file contained 2 of these alerts with 2 unique sources to 2 unique destinations. All the packets are from internal IP addresses destined for external IP addresses with the destination port 80. The two destination addresses are **xxx.yyy.98.157** and **xxx.yyy.97.66**. Both addresses were alerted on other numerous other attacks.

The xxx.yyy.98.157 has a total of 44 entires in the alerts log. It shows xxx.yyy.98.157 was involved in SMB Name Wildcard, ISAPI Overflow and portscans as follows.

```
[**] SMB Name Wildcard [**] 62.11.88.61:1026 -> xxx.yyy.98.157:137
[**] SMB Name Wildcard [**] 65.27.128.245:1030 -> xxx.yyy.98.157:137
[**] IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize [**] xxx.yyy.98.157:2580 -> 130.158.70.194:80
[**] NIMDA - Attempt to execute cmd from campus host [**] xxx.yyy.98.157:2706 -> 130.158.163.182:80
[**] spp_portscan: PORTSCAN DETECTED from xxx.yyy.98.157 (THRESHOLD 12 connections exceeded in 4
seconds) [**]
 [**] SMB Name Wildcard [**] 203.243.242.131:1026 -> xxx.yyy.98.157:137
```

The xxx.yyy.97.66 has a total of 3243 entries in the alerts log. It shows xxx.yyy.98.157 was involved in SMB Name Wildcard, ISAPI Overflow and portscans as well. I have not included the log entries due to the large number.

Recommendation: Review logs to see what other systems were attacked and attempt to determine when it was infected. Follow the cert advisory CA-2001-26 to recover, contain and eradicate the worm.

References: http://www.cert.org/advisories/CA-2001-26.html.

4.47 EXPLOIT digital unix noop

The alerts file contained 2 of these alerts with 1 unique source to 1 unique destination. The packet is from one external IP address destined for an internal IP address with the destination port 20. The alert is trigger based on content.

```
130.94.149.162:20 -> xxx.yyy.24.47 4519
```

Recommendations: Since port 20 is used by FTP for file transfers, these are probable false positives. Capture packers to analyze the content.

4.48 Trin00 password on tcp

The alerts file contained 1 of this alert with 1 unique source to 1 unique destination. The packet is from an external IP address destined for an internal IP address with the source port 80 and the destination port 3382. I could not locate the snort rule on the default configuration. Assume it is triggering on the content of the packet.

```
209.100.212.5:80 -> xxx.yyy.168.168:3382
```

No other suspicious activity was logged from the source or destination.

Recommendation: Capture the packet to analysis the content, inpect xxx.yy.168.168 for Trin00.

4.49 External FTP to HelpDesk xxx.yyy.53.29

The alerts file contained 1 of this alert with 1 unique source to 1 unique destination. The packet was from an external IP address to the internal IP address xxx.yyy.53.29 with the destination port 21. The one source is registered to as shown below.

```
213.245.23.0 - 213.245.23.255
MEULUN-CABLE
Chello France
Meulun
Private customer Cablemodems
FR
```

Recommendations: Block access to all internal FTP systems unless required.

4.50 DDOS shaft client to handler

The alerts file contained 1 of this alert with 1 unique source to 1 unique destination. The packet is from an external IP address destined for an internal IP address with the source port 80.

```
213.219.122.10:80 -> xxx.yyy.221.22:20432
```

Rule that triggered it is

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 20432 (msg:"DDOS shaft client to
handler"; flags: A+; reference:arachnids,254; classtype:attempted-dos;
sid:230; rev:1;)
```

Recommendations: Capture packet for further analysis.

4.51 Bugbear@MM virus in SMTP

The alerts file contained 1 of this alert with 1 unique source to 1 unique destination. The packet is from an external IP address destined for internal IP address with source port 25.

```
129.78.64.15:58040 -> xxx.yyy.6.47:25
```

Recommendation: Capture packets to analyze the contents. Scan internal system for backdoor ports. If this is an SMTP host assure that it is protected by an Anti-Virus application with current signatures.

Reference: www.itc.virginia.edu/desktop/virus/results.php3?virusID=53

4.52 Back Orifice

The alerts file contained 1 of this alert with 1 unique source to 1 unique destination. The packet is from an external IP addresses destined for an internal IP addresses with destination port of 31337. Since 31337 is an ephemeral port if may be a false positive.

```
63.250.207.64:61610 -> xxx.yyy.88.154:31337
```

Recommendations: Capture packets to analyze the content to ensure it an attack and not a false positive. Inspect 63.250.207.64 to see if it contains Back Orifice.

5. Top talkers list in terms of scans alerts and OOS and altogether.

Top ten external source addresses from alerts file, based on number of packets.

Number of packets	Source IP address
9896	212.179.101.68
9111	212.179.48.2
5389	66.42.68.210
2383	212.179.102.138
2288	24.66.182.171
2063	4.46.32.83
1812	62.65.192.30
1783	212.179.85.46
1745	207.46.134.190
1533	212.179.35.118

Top ten external sources address from the scans file, based on number of packets.

Number of packets	Source IP address
32155	218.68.216.47
6992	172.186.87.8
5655	217.21.114.148
5235	217.21.114.154
4049	63.250.195.10
3543	216.173.52.200
2835	217.59.215.194

2653	61.133.3.146
2076	64.5.44.143
1795	62.65.192.30

Top ten external sources address from the oos file, based on number of packets.

Number of packets	Source IP address
1099	68.54.93.181
612	209.191.132.40
356	66.140.25.157
162	148.63.151.3
159	61.114.222.241
152	62.142.15.248
147	212.244.86.66
136	80.26.5.150
136	216.95.201.23
119	216.95.201.34
117	69.3.109.174

6. List of 5 external IP address and registration information.

These 5 were chosen because of their foreign registration and hostile activity directed at the UNIVERSITY network.

6.1 Bezeq International

The address range, 212.179.192.0 - 212.179.255.255, is registered to the BEZEQINT, HOSTMASTERS TEAM, bezeq-international, 40 hashacham address, petach tikva, 49170, Israel.

This address range triggered 39,363 alerts due to a rule that watches for this address range. In addition to these 39.363 alerts, 212.179 showed 142 other alerts. This consisted of a NMAP scan, 3 possible red worm, and a CS WEBSERVER - external web traffic alert. A search for address with 212.79 in the oos file showed no packets. A search in the scans file showed

```
Apr  5 09:37:57 xxx.yyy.194.223:55909 -> 212.179.201.233:5062 SYN ******S*
Apr  5 09:52:45 xxx.yyy.194.223:60754 -> 212.179.201.233:5062 SYN ******S*
Apr  5 09:44:52 xxx.yyy.194.223:57996 -> 212.179.201.233:5062 SYN ******S*
Apr  5 15:28:52 xxx.yyy.194.247:13139 -> 212.179.220.217:13139 UDP
```

I could not find a reference for port 5062 but 13139 is used by Gamespy software. This could be used by a foreign host to gain a vector of attack to collect information from the UNIVERSITY systems.

6.2 Chinese Academy of Sciences

The address range, 159.226.0.0 -159.226.0.0, is registered to the The Computer Network Center Chinese Academy of Sciences, P.O. Box 2704-10, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China

This address range triggered 1,254 alerts due to a rule that watches for this address range. In addition to these 1,254 alerts, 212.179 showed 17 other alerts. This consisted of SMB Wildcard, but there was a Queso, and a CS WEBSERVER - external web traffic alert. A search for address with 159.226.0.0 in the oos130 and scans_130 14 packets showed.

```
oos130:04/01-22:02:20.111923 159.226.113.140:44755 -> xxx.yyy.130.14:80
oos130:04/01-22:47:18.817356 159.226.113.140:46066 -> xxx.yyy.130.14:80
oos130:04/02-02:35:54.042964 159.226.113.140:53650 -> xxx.yyy.130.14:80
oos130:04/02-02:36:02.562933 159.226.113.140:53651 -> xxx.yyy.130.14:80
oos130:04/02-04:33:49.026361 159.226.113.140:57370 -> xxx.yyy.130.14:80
oos130:04/02-07:12:29.094078 159.226.162.168:1357 -> xxx.yyy.100.237:9080
scans_130:Apr  2 02:36:02 159.226.113.140:53651 -> xxx.yyy.130.14:80 SYN 12****S* RESERVEDBITS
scans_130:Apr  2 05:16:17 xxx.yyy.97.88:3915 -> 159.226.155.101:80 SYN ******S*
scans_130:Apr  4 04:03:06 xxx.yyy.1.3:32790 -> 159.226.1.1:53 UDP
scans_130:Apr  4 04:03:11 xxx.yyy.1.3:32790 -> 159.226.118.1:53 UDP
scans_130:Apr  4 04:03:21 xxx.yyy.1.3:32790 -> 159.226.1.1:53 UDP
scans_130:Apr  4 04:03:22 xxx.yyy.1.3:32790 -> 159.226.1.1:53 UDP
scans_130:Apr  4 04:03:25 xxx.yyy.1.3:32790 -> 159.226.1.1:53 UDP
scans_130:Apr  5 02:55:51 xxx.yyy.137.7:29460 -> 159.226.1.1:53 UDP
```

References: http://ws.arin.net/cgi-bin/whois.pl

6.3 HanWang Technology Co.LTD

The address 210.160.200.2 performed a NETBIOS null scan of 6 different internal systems. The address 210.160.200.2 also alerted on a rule that watches for traffic to xxx.yyy.30.3. It appears to be a NETBIOS null scan as well.

The address range, 210.160.200.0 -210.160.203.255 , is registered to the HanWang Technology Co.LTD, BeiJing, China.

6.4 Estonia Telephone Co

The address 213.219.90.74 performed a SYN-FIN scan and was listed for over 200 other alerts.

The address range, 213.219.89.0 - 213.219.90.255 , is registered to the Estonian Telephone Co/Estpak Data Ltd., Sole str 14, Tallinn, Estonia.

6.5 MEULUN-CABLE

The address 213.245.23.68 performed a SYN scan of 1,323 internal addresses and attempted a FTP to the HelpDesk.

External FTP to HelpDesk xxx.yyy.53.29 [**] 213.245.23.68:4101 -> xxx.yyy.53.29:21

The address range, 213.245.23.0 – 213.245.23.255, is registered to MEULUN-CABLE, Chello France, UPC Technology, Internet Services, Erlachplatz 116, A-1100, Vienna, Austria

7. Correlation from previous practical, GCIA #0209 and above.

Correlations to specific alerts are listed under the specific alert under the references section. The correlation of recommendations and conclusions are as follows.

John Melvin analyzed logs from October 14th through Oct 18th, 2002. He agreed with my conclusion that an ACL should be setup on the border routers and the IDS sensors be tuned.

www.giac.org/practical/GCIA/John_Melvin_GCIA.pdf

Fred Thiele analyzed logs from November 27th through December 1st, 2002. He concurs that a strict security policy needs to be in place with firewalls rules that only allow services that are in use should be permitted to pass through the firewall.

www.giac.org/practical/GCIA/Fred_Thiele_GCIA.pdf

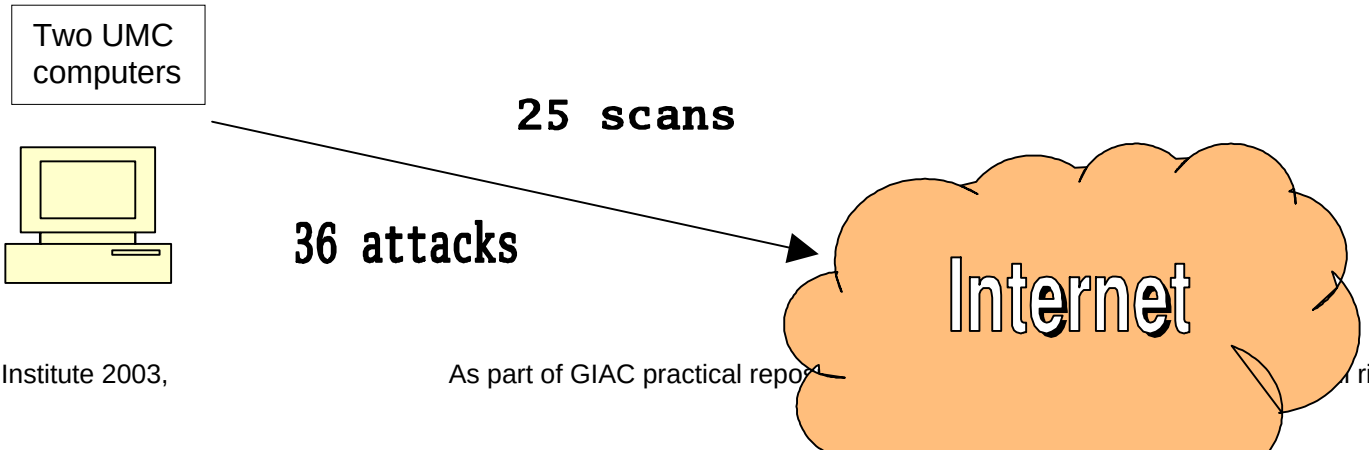
Donald Gregory analyzed logs from August 1st through August 5th, 2002. He concurs in his security recommendations section that the IDS sensor is noisy and the firewall should disallow unauthorized hosts and service.

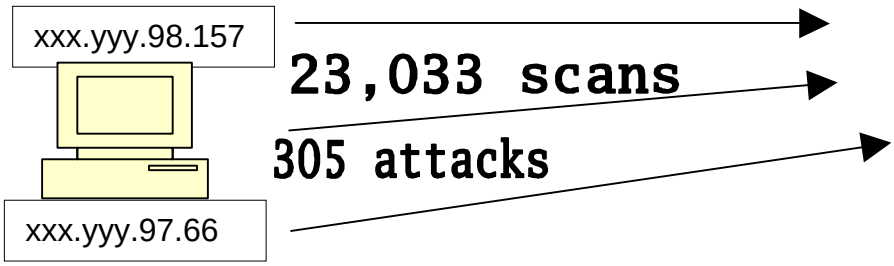
www.giac.org/practical/GCIA/Donald_Gregory_GCIA.pdf

Further correlations can be found in the references section of the detail analysis in section 4.

8. A link graph of some portion of the data file to show relationship.

The link graph shows hostile activity from two UNIVERSITY systems infected with the NIMDAS virus. Specifically, the internal xxx.yyy.98.157 scanned 25 systems on the Internet and performed 36 attacks. The xxx.yyy.97.66 scanned 23,033 scans and 305 attacks. Since they systems are infected with they are probably attacking internal UNIVERSITY systems as well but are not logged based on the placement of the sensor.





9. Insight

Four internal systems, xxx.yyy.97.66, xxx.yyy.98.157, xxx.yyy.97.88 and xxx.yyy.98.184, appear to be infected with the NIMDA virus. Other systems may be compromised as well but not enough activity is recorded to substantiate this conclusion. A review of the content should be performed.

Internal systems are using Mircosoft's IP address to send packets to the external network. Using somebody else's IP address is usually a sign of malicious activity.

Packets with previously reserved ECN bits are being captured in the oos when it appears they are only using the new congestion notification technique.

Content needs to be captured and reviewed.

Firewall/router filtering policy is inadequate to protect the UNIVERSITY network. The SMB wildcards, external to external and IDS systems are logging many false positives. Tuning needs to be performed to reduce the amount of alerts.

10. Defensive recommendations

The UNIVERSITY network needs to adopt a defense in depth posture. Policy, Intrusion Detect Sensors, Firewalls, Access control lists and virus scanning needs to be a combined to create multilevel of defense from outside attacks as well as inside attacks.

The Policy of the UNIVERSITY network is not apparent from the alerts and scans but it is apparent that there is a lot of P2P activity, addresses being spoofed and other questionable activity inside the UNIVERSITY network. If not already done, a clear policy should be adopted and distributed to users of the network. Violations of the policy should result in denial of the UNIVERSITY network access.

The IDS picks up way too much traffic to make timely response possible. Many of the alerts can be logged instead alerting, for later investigation. Much of the traffic should be blocked at the border router or internal firewall. Content needs to be captured as well to be able to analyze whether the alert is a false positive. The active NIMDA and other suspicious activity

in the network may warrant the placement of additional IDS devices inside the network. It may also be helpful to have IDS on the inside of the firewall to see what got through.

The firewall appears to be allowing internal devices to send out packets with private address and obviously spoofed address. It appears the firewall is blocking specific ports since a now of the internal web servers seemed to be under attack from the Internet. A stateful firewall or proxy based should be used to enhance the ability to block scans.

The access control list on the Internet facing router should be blocking all NETBIOS traffic. This would eliminate the flood of alerts on the IDS system. Access lists can also be used to block foreign addresses that continue to exhibit hostile activity.

The virus software appears to be marginally effective since only four machines are showing strong signs of an active NIMDA and Red Worm infection. Many others systems are involved in suspicious activity that should be investigated since NIMDA and Red Worm have multiply vectors for attack that become available for propagating the virus when the virus is introduced into a network.

A final recommendation would be to create security zones within the network to segregate the students from the servers and other valuable assets as well as separating the public from the private systems.

11. Description of your analysis process.

The steps I went through in my analysis process.

11.1 Retrieving files, reviewing format, determining number of events and validating data.

I download the three sets of files for each of the days from April 1st to April 5th. I then concatenated the same file types to get all 5 days in one file. I viewed the files to see what type of data was in each file.

11.1.1 The scans file.

The scans file appears to be the output of the portscan preprocessor.
The scans contain one-line entries. The format is

```
Month Day time sourceIP:sourceport -> destinationIP:destinationport protocol
```

Sample as follows:

```
Apr 1 00:46:44 xxx.yyy.210.182:14567 -> 12.224.147.97:3522 UDP
```

Since each entry consisted of one line, running it through **'wc -l'** gave me 1,416,564 entries. A quick look at the file contents shows that all the entries seem to be with the UDP protocol. I used **`grep UDP scans -c`** to find the number of lines in the file where UDP is 1,287,936, a difference of 128,628 packets. I grepped the scans file with the -v option to extract lines

without UDP **'grep UDP scans -v > scansnoUDP'**. The file contains 128,628 entries. Most of them appear to be packets with bad TCP flag combinations. The UDP scan entries are ten times greater than the TCP scan entries.

11.1.2 The alerts file.

The alert file appears to be the output of Snort -A fast option. The alerts file also contains one line per entry. The format is

```
DateTime [**] alert description [**} sourceIP:sourceport -> destIP:destport
```

Sample as follows:

```
04/01-00:00:11.222529  [**] TFTP - Internal TCP connection to
external tftp server [**] 205.188.11.236:69 -> MY.NET.240.94:4976
```

Since each entry consists of one line, running it through **'wc -l'** gave me 317,653 lines.

11.1.3 The OOS file.

The OOS file contains header information of each packet as well as the hex and ASCII decodes of the packet header and beginning of the payload data. The first packet in the file OOS_Report_2003_04_01_21757 had a timestamp of 03/31-00:06:09. Further investigation showed that all the OOS files were from the day before the name of the file indicated except for the 5th. The OOS data for the 5th was contain in a file label indicating April the 7th. This showed that the file label indicates the day or days after the capture was taken, not the day of capture. The amount of data for each packet varies depending on the amount of data in the payload. The format of the header of each packet is

First line

```
DateTime sourceIP:sourceport -> destinationIP:destinationport
```

Second line

```
TCP TTL:(time to live value)  TOS:(Type of Service Value) ID:(TCP
packet ID) IpLen:(Ip header length) DgmLen:(datagram length)
Fragmentation  bits
```

Third line

```
(TCP bits) Seq: (TCP sequence number)  Ack: (TCP packets
acknowledged)  Win: (windos size)  TcpLen: (TCP header length)
```

Fourth line

```
TCP Options (number of TCP options) => (TCP options)
```

Sample as follows

```
04/01-01:09:34.063825 64.71.184.56:44646 -> MY.NET.24.22:25
TCP TTL:51 TOS:0x0 ID:11035 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xCB5CCA05  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 1536750246 0 NOP WS: 0
```

Each packet is separated with a line of repeating =+=+=+=+. A grep of the combined OOS files for the +=+=+=+=+= string '**`grep +=+=+=+= oos -c`**' plus one gave me 9,400 lines.

As noted in the following correlation section. The oos file was analyzed separate from the alerts and scans file for interesting traffic then correlated to alerts and scans.

11.2 Home network identification

This section of the analysis has been deleted since it contains a methodology that can be used to determine the identity of the University that supplied log files.

11.3 Packet Correlation

In order to match the scans file to the oos and alerts I needed to either change the my.net in the alerts and oos to xxx.yyy or change the scans from xxx.yyy to my.net. Since the scans file is much bigger than the oos and alerts combined I decided to change the alerts and oos files. I did this by running the files through sed replacing my.net with xxx.yyy.

```
cat alerts | sed s/MY.NET/xxx.yyy/g > alerts1xxx and cat oos | sed
s/MY.NET/xxx.yyy/g > oosxxx
```

I verified all instances of MY.NET were replace by grepping for the MY.NET sting in both files.

Now that I had the files cleaned up and data consistent I devised the theory that some of the scans data would show up in the alerts file and the oos entries would appear in the scans file TCP entries. I extracted the UDP scan source addresses,

```
grep UDP scans | cut -f4 -d ' '| sed "s/\:/ /" | cut -f1 -d ' ' | sort
| uniq -c  > scansUDPsource
```

and then grepped for some of these addresses from the scans in the alerts file. Several matches were found as follows.

```
04/04-03:29:02.872654  [**] spp_portscan: portscan status from
xxx.yyy.132.23: 90 connections across 1 hosts: TCP(0), UDP(90) [**]
04/04-03:29:03.109355  [**] spp_portscan: portscan status from
xxx.yyy.132.23: 130 connections across 1 hosts: TCP(0), UDP(130) [**].
```

This makes sense since the spp_portscan is what made the entries in the scans log. Now I did the same thing for the alert and oos files. Some matched as follows.

Alert file matches.

```
04/01-01:23:26.116013  [**] Queso fingerprint [**]
66.140.25.157:34384 -> xxx.yyy.60.16:8000
```

oos file matches.

```
04/01-01:23:26.116017 66.140.25.157:34384 -> xxx.yyy.60.16:8000
```

But other entries in the oos file did not match in the alerts files. The oos file contained many KaZaa packet captures, which did not have a corresponding entry in the alerts file as shown below.

oos file without matches.

```
04/01-00:52:37.376980 148.63.155.220:2487 -> xxx.yyy.237.114:1382
TCP TTL:112 TOS:0x0 ID:26929 IpLen:20 DgmLen:444 DF
****P*** Seq: 0x168700A  Ack: 0x0  Win: 0x2000  TcpLen: 20
47 45 54 20 2F 2E 68 61 73 68 3D 39 34 63 62 39       GET /.hash=94cb9
38 34 63 30 30 36 34 63 65 37 30 36 62 64 64 32       84c0064ce706bdd2
64 36 33 63 31 65 33 62 62 36 63 66 63 64 32 33       d63c1e3bb6cfcd23
36 38 34 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F       684 HTTP/1.1..Ho
73 xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx       st: xxx.yyy.237.1
31 34 3A 31 33 38 32 0D 0A 55 73 65 72 41 67 65       14:1382..UserAge
6E 74 3A 20 4B 61 7A 61 61 43 6C 69 65 6E 74 20       nt: KazaaClient
```

Based on this I decided to analyze the alerts file with the scans file since it appeared to be a summary of the scans and examine the oos on its own. The detailed alerts analysis includes packets with the same source address that was found in the scans file. The oos file was analyzed separately.

11.4 Analysis of Alerts

I extracted the alerts from the alerts files and sort, removed duplicates and resorted to get a count of how many of each alert.

```
cat alertsxxx | cut -f4- -d' ' | cut -f1 -d'[' | sort | uniq -c | sort -r > alertsSummary
```

Unfortunately the portscan lists the IP address in the alert message so I received 14776 unique alerts. To fix this I extracted all the lines without the spp_portscan in the alert message.

```
grep spp_portscan alerts130 -v > alertsNOscan
```

Then ran it through the extract again.

```
cat alertsxxxNOscan | cut -f4- -d' ' | cut -f1 -d'[' | sort | uniq -c | sort -r > alertsSummary
```

A quick wc -l on the output file shows 76 lines, much better. A look through the output filed showed some junk as follows

```
1 :2489
1 -> 218.50.53.17:137
1 :2092
```

It appears that there are some parts of lines in the alerts file as were in the scans file. Most appear to be the last of the entry so to clean up the file I extracted all the lines that contained a "[".

```
cp alertsxxxNOscan alertsxxxNOscanold
grep '\[' alertsxxxNOscanold > alertsxxxNOscan
```

Then ran it through the extract again.

```
cat alertsxxxNOscan | cut -f4- -d' ' | cut -f1 -d'[' | sort | uniq -c | sort -r > alertsSummary
```

Another wc – on the output file showed 53 lines. More improvement.

I then ran the alerts file through SnortSnarf. After 26 hours SnortSnarf was still running. I aborted the program and reduce the alerts file by removing all the SMB alerts. I used the SnortSnarf output to in the analysis of all the alerts with the exception of the SMB alerts.

A complete analysis is shown under section 2 along with a brief description.

11.5 Analysis of OOS file.

The combined oos file contains 9399 entries of which 8558 have reserved bits in the flag field set with the syn bit (12****S*). The reserved bits can now be used with the syn bit to contain the flag combination 12****S*. This is defined in rfc3168 . The complete packet header follows.

```
04/01-00:13:34.307707 216.99.199.78:36847 -> xxx.yyy.202.214:6346
TCP TTL:46 TOS:0x0 ID:8300 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xE2B6F842  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 361683285 0 NOP WS: 0
```

These packets are probably false positives and will be deleted from further analysis. I deleted the entries by writing a perl script.

I then used the following string of commands to determine what other flag combinations were present and how many different combinations occurred.

```
grep Seq oosNOecnpl | cut -f1 -d'| sort | uniq -c | sort -n
grep Seq oosNOecnpl | cut -f1 -d'| sort | uniq -c | sort | wc -l
```

There were 85 different flag combinations. With all these variations of packets, I needed to find a list of legal/illegal flag combinations. The following is a list of the legal flags and the

illegal flag combinations. I derived this from the website http://www.securityfocus.com/infocus/1200.

List of flag rules.

1. At least one of these six flags must be set in each TCP packet; each flag corresponds to a particular bit in the TCP header.

2. Except for the initial SYN packet, every packet in a connection must have the ACK bit set.

3. SYN FIN is probably the best known illegal combination. Remember that SYN is used to start a connection, while FIN is used to end an existing connection. It is nonsensical to perform both actions at the same time. Many scanning tools use SYN FIN packets, because many intrusion detection systems did not catch these in the past, although most do so now. You can safely assume that any SYN FIN packets you see are malicious.

4. SYN FIN PSH, SYN FIN RST, SYN FIN RST PSH, and other variants on SYN FIN also exist. These packets may be used by attackers who are aware that intrusion detection systems may be looking for packets with just the SYN and FIN bits set, not additional bits set. Again, these are clearly malicious.

5. Packets should never contain just a FIN flag. FIN packets are frequently used for port scans, network mapping and other stealth activities.

6. Some packets have absolutely no flags set at all; these are referred to as "null" packets. It is illegal to have a packet with no flags set.

Some other bad flag combinations where found at http://www.linuxhelp.net/guides/iptables/

IPTables Firewall Script except

7. FIN,URG,PSH
8. SYN,RST,ACK,FIN,URG
9. SYN,RST
10. ECN bits can be used with any combination of other flags that are legal.

I also needed a explanation on how the ECN bits work. Extracts from the website ftp://ftp.isi.edu/in-notes/rfc3168.txt explain the use of the ECN bits in the following.

> "Explicit congestion Notification (ECN) is performed in the IP header but requires the support of the transport layer. This is provided with the use of two of the TCP flag bits previously reserved. The new flag bits are Congestion Window Reduced (CWR) and ECN-echo (ECE). The new flag bit order is CWR, ECE, URG, ACK, PSH, RST, SYN and FIN respectivly. The new flag fields have very little to do with the other flags and where they do they can occur in any combination."

With the list of flag combinations, criteria for non-ECN legal and illegal flag combinations, and a explanation of ECN flag bits, I created the a list with the number of occurrences, the different flag combinations and the possible reason they are flagged.

I then reduced this list by rewriting my perl script to remove the entries that seemed to trip for just the ECN flags being set as denoted by rule10. This left 70 different flag combinations and 871 packets. In reviewing the perl script output I noticed most of the packets were from Kazaa using Gnutella software with the flag pattern ****P***. This appears to be legitimate traffic with poor TCP code. To confirm all the ****P*** packets were of this type I modified the perl script again to extract all packets with the ****P*** flag combination. I ran following shell commands script to collect the source IP addresses and number of occurrences.

```
grep '\->' kz | cut -d' '  -f2 | sed "s/\./ /2"  | cut -d' ' -f1 | sort | uniq -c | sort -nr
```

The output follows.

```
315 148.63
 95 148.64
 30 64.86
 28 200.167
  2 203.177
  1 4.63
```

I modified the perl script to delete these entries as well since they appear to be legitimate traffic although many Gnulleta clients install spyware and create a possible vector for attack. This left 340 packets with 69 different flag combinations. There are 124 different source addresses with eight internal addresses and 147 different destination addresses with 103 internal addresses.

I then combined the two lists and deleted the internal address. I used the list as input for search stings with grep for external addresses and ran it against the alerts file.

```
grep '\->' oosnokz | cut -d' '  -f2 | sed "s/\:/ /" | cut -d' ' -
f1 | sort | uniq  > oosaddresses
grep -foosaddresses alertsxxx
```

This matched with 278 packets. Out of the 278 packets 264 were related to a scan which was triggered due to the illegal flag combination. The remaining 14 packets are as follows.

```
EXPLOIT x86 setgid 0 [**] 168.143.179.114:80 -> xxx.yyy.204.110:4815
Queso fingerprint [**] 216.95.201.39:52122 -> xxx.yyy.24.22:25
spp_http_decode: IIS Unicode attack detected [**] xxx.yyy.153.112:4346 -> 168.143.179.114:80
spp_http_decode: IIS Unicode attack detected [**] xxx.yyy.153.112:4346 -> 168.143.179.114:80
spp_http_decode: IIS Unicode attack detected [**] xxx.yyy.153.112:4346 -> 168.143.179.114:80
CS WEBSERVER - external ftp traffic [**] 213.156.61.154:3629 -> xxx.yyy.100.165:21
Queso fingerprint [**] 216.95.201.39:37173 -> xxx.yyy.6.40:25
Probable NMAP fingerprint attempt [**] 66.96.222.130:50684 -> xxx.yyy.70.107:23
NMAP TCP ping! [**] 66.96.222.130:50687 -> xxx.yyy.70.107:1
Queso fingerprint [**] 216.95.201.39:37525 -> xxx.yyy.24.22:25
Probable NMAP fingerprint attempt [**] 67.41.169.248:17105 -> xxx.yyy.9.11:80
SUNRPC highport access! [**] 67.41.169.248:12164 -> xxx.yyy.9.11:32771
```

CS WEBSERVER - external web traffic [**] 68.33.106.217:1723 -> xxx.yyy.100.165:80
spp_http_decode: CGI Null Byte attack detected [**] xxx.yyy.97.45:1294 -> 168.143.179.114:80

The NMAP and Queso alerts were also triggered due to illegal flag combinations so I deleted them as well as the packets from external devices which left me with the following.

spp_http_decode: IIS Unicode attack detected [**] xxx.yyy.153.112:4346 -> 168.143.179.114:80
spp_http_decode: CGI Null Byte attack detected [**] xxx.yyy.97.45:1294 -> 168.143.179.114:80

I include these results under each of the appropriate detailed alerts section to indicate that they may have been successfully compromised by an external system.

11.6 Analysis of Scan File

Since being scanned from devices from the Internet appears to be a part of being on the Internet, I focused my attention to extracting the internal addresses that are scanning. I then correlated this to the alerts file to determine if the internal address are involved in any other suspicious activity.

I used the command string to extract the source internal IP addresses with the following command.

```
cut -d' ' -f5 scans_130 | sed 's/:/ /' | cut -d' ' -f1 |
grep xxx.yyy | sort | uniq > scans_interal
```

The output file, scans_interal, contained 254 entries. I used the scans_interal as the pattern match file to input into grep to extract alerts with any of these internal addresses.

```
grep -fscans_interal alerts > alerts-scans.
```

The output file contained 89,519 entries. I then deleted the timestamps to better summarize the data.

```
cut -d' ' -f2- alerts-scans > alerts_scans2
```

Since the previous grep also matched on internal address as the destination I reduced the alerts_scans2 file by greping for the pattern '] xxx.yyy" to get just the packets with the internal source address. Since the alerts from the portscans does not contian this string all the portscan alerts were elimated as well.

grep '] xxx.yyy' alerts_scans2 > alerts_scans_src, entries 14,152

I wanted to see how many alerts I was left with so I ran the following command to give me all the unique alerts in the alerts_scans_src.

```
cat alerts_scans_src | cut -d']' -f2 | sort | uniq
```

The following is a list of the alerts I had left.

xxx.yyy.30.4 activity [**
High port 65535 tcp - possible Red Worm - traffic [**
High port 65535 udp - possible Red Worm - traffic [**
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize [**
Incomplete Packet Fragments Discarded [**
IRC evil - running XDCC [**

NIMDA - Attempt to execute cmd from campus host [**
NIMDA - Attempt to execute root from campus host [**
Possible trojan server activity [**
SMB Name Wildcard [**
spp_http_decode: CGI Null Byte attack detected [**
spp_http_decode: IIS Unicode attack detected [**
TCP SRC and DST outside network [**
TFTP - Internal TCP connection to external tftp server [**
TFTP - Internal UDP connection to external tftp server [**
Tiny Fragments - Possible Hostile Activity [**
Watchlist 000220 IL-ISDNNET-990517 [**

Next I wanted to eliminate the destination IP addresses and source ports so I could summarize the source addresses with the alerts. I tried to use the ":" as a delimiter in the cut command but the spp_http_decode contained a ":" in the alert description so I extracted the spp_http_decodes with the following commands while removing the destination IP address and source port.

```
grep  spp_http_decode alerts_scans_src | cut -d':' -f1-2 > spp_http_decode
```

I then eliminated the spp_http_decode from the rest of the alerts while removing the destination IP address and source port.

```
grep  spp_http_decode -v alerts_scans_src | cut -d':' -f1 > no_spp_http
```

The tiny fragments alerts doesn't include a source port so the ":" was missing thus they still contained the destination addresses. To fix this I extracted the tiny fragments to another file and removed the destination address using the "-" delimiter.

```
grep Tiny  no_spp_http | cut -d'-' -f1-2 > tiny_no_src
```

I then removed the tiny fragments from the rest of the alerts file.

```
grep Tiny -v  no_spp_http > no_tiny_alerts
```

Now I combined the three files back together using the following cat command.

```
cat spp_http_decode tiny_no_src no_tiny_alerts > combined_no_src
```

I then sorted and list all the unique entries to determine what systems were scanning as well as triggering other alerts.

```
sort combined_no_src | uniq > comb_uniq
```

The comb_uniq file contains 220 entries.

I included the source IP address under each of the alerts in the detail alerts section stating these internal addresses also participated in scanning activity as follows,

References:

archives.neohapsis.com/archives/sf/ids/2002-q2/0018.html
archives.neohapsis.com/archives/snort/2000-01/0222.html
archives.neohapsis.com/archives/snort/2000-05/0115.html
archives.neohapsis.com/archives/snort/2000-11/0244.html
cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-0138
h30097.www3.hp.com/demos/ossc/doc/snort-1.8p1/RULES.SAMPLE
isc.incidents.org/port_details.html?port=13139
lists.suse.com/archive/suse-security/2001-Mar/0371.html
lists.suse.com/archive/suse-security/2002-Feb/0374.html
ws.arin.net/cgi-bin/whois.pl
www.ciac.org/ciac/bulletins/k-037.shtml
www.cipherdyne.com/cgi/viewcvs.cgi/psad/psad_signatures.diff?r1=1.19&r2=1.20
www.cipherdyne.com/cgi/viewcvs.cgi/psad/psad_signatures.diff?r1=1.6&r2=1.7
www.europe.f-secure.com/v-descs/adore.shtml
www.geocrawler.com/archives/3/90/2001/9/0/6722577/
www.giac.org/practical/GCIA/Donald_Gregory_GCIA.pdf
www.giac.org/practical/Edward_Peck_GCIA.doc
www.giac.org/practical/GCIA/Fred_Thiele_GCIA.pdf
www.giac.org/practical/GCIA/John_Melvin_GCIA.pdf
www.giac.org/practical/Mike_Bell_GCIA.doc
www.giac.org/practical/GCIA/Susan_Kovacevich_GCIA.pdf
www.giac.org/practical/Tod_Beardsley_GCIA.doc
www.iana.org/assignments/port-numbers
www.icir.org/floyd/papers/rfc3168.txt
www.itc.virginia.edu/desktop/virus/results.php3?virusID=53
www.portsdb.org/bin/portsdb.cgi?portnumber=161&protocol=ANY
www.portsdb.org/bin/portsdb.cgi?portnumber=515&protocol=ANY
www.portsdb.org/bin/portsdb.cgi?portnumber=69&protocol=ANY
www.russonline.net/tonikgin/EduHacking.html
www.sans.org/rr/casestudies/outbound.php
www.sans.org/rr/malicious/code_red8.php
www.sans.org/rr/malicious/nimda3.php
www.sans.org/y2k/subseven.htm
www.securiteam.com/exploits/5JP0A1P9GQ.html
www.silicondefense.com/software/snortsnarf
www.snort.org/docs/writing_rules/chap1.html - tth_sEc1.3
www.snort.org/docs/writing_rules/chap2.html#tth_sEc2.4.1
www.snort.org/docs/writing_rules/chap2.html - tth_sEc2.4.2
www.snort.org/snort-db/sid.html?sid=522
www.symantec.com
www.UNIVERSITY.edu/engineer/cse
www.whitehats.com/IDS/552
www.xfocus.net/exploits/linux_ntpd.txt
ftp://ftp.isi.edu/in-notes/rfc3168.txt
http://www.cert.org/incident_notes/IN-2000-03.html
http://www.cert.org/advisories/CA-2001-26.html