Global Information Assurance Certification Paper

A Method for Tamper Detection on Secure Local Networks

SANS/GSEC Practical Assignment
Security Essentials Certification

Author: **Richard Chadderton**
Assignment Version: **1.4b (Option 1)**
Submitted: **Wednesday, February 04, 2004**

Table of Contents

Abstract

Operators of secure local networks use several methods to increase confidence in the integrity of their physical infrastructure. Physical security and access controls are vital, as are process controls for adds, moves, and changes to the network infrastructure. Logical network monitoring is a very important component of this. Logical controls monitor the network topology and can help identify unauthorized changes.

This paper describes a specific logical control method referred to here as port-status monitoring. It functions alongside other established logical control methods such as IP Subnet Probing, Ethernet Protocol Analysis, and ARP Monitoring. It focuses specifically on monitoring the state of connections in Ethernet networks where confidence in the security and integrity of a local area network is paramount.

Discussion of Methods

This research paper examines a method for increasing confidence in the integrity of highly secure private local Ethernet networks. This method, referred to here as port status monitoring, leverages the reporting capability of Cisco's Catalyst Ethernet switching devices to identify changes in the status of local ports.[1]

It is possible to set up a system where these status messages can be covertly collected, parsed, and analyzed for indications of tampering, or other loss of integrity. The goal of this method is to be able to track when the Ethernet connection of protected node is disconnected or otherwise tampered with.

Some of the ideas behind this method came from the author's own unpublished work in anti-tampering technology during 2000 and 2001, and stem from a need to initiate a timely security response when certain security devices were tampered with. Additional ideas on automated log file analysis were developed by Networking Unlimited Inc.[2] The components used in preparing a proof-of-concept of this method are readily available from public internet sources.

The reasons for pursuing this approach to network integrity is that the traditional methods of integrity validation (i.e.: IP-level scanning and probing, manual protocol and traffic analysis, and Ethernet address monitoring) have limits. Although these are useful, and perhaps even necessary in secure private networks, they are unable to reliably detect temporary interruptions in the physical path, such as what might be observed during the installation of an intrusive network tap device, or the unauthorized attachment of a rogue workstation.

IP-Level Scanning

An IP-level scanning tool such as NMAP (http://www.insecure.org/nmap/) is extremely useful in discovering hosts and mapping IP protocols and services, and should generally be employed in all secure network installations. However it is designed primarily to interact with IP, and is not particularly useful for non-IP applications. It relies upon the target's adherence to IP standards when it makes assumptions about target system behavior. Mark Wolfgang describes in his paper "**Host Discovery with nmap**[3]" the different ways NMAP can interrogate a target IP stack and how those can be defeated

---

[1] Cisco. Catalyst 3550 Multilayer Switch Software Configuration Guide. (San Jose, CA: Cisco Systems Inc. Dec 2002.) pp26.1 – 26.12.

[2] Jones, Dr. Vincent C. "Automated Analysis of Cisco Log Files." Networkingumlimited.com. 1999 URL: http://www.networkingunlimited.com/white007.html (4 Feb 2004)

[3] Wolfgang, Mark. "Host Discovery with nmap.", Moonpie.org. Nov. 2002. URL: http://moonpie.org/writings/discovery.pdf (3 Feb. 2004)

with network filters and firewalls. He suggests that "nmap's default discovery method is insufficient" when the target device is protected with a very specific firewall.[4]

NMAP can also be confused by a rouge system that intentionally provides misinformation. In his paper entitled "**A practical approach for defeating Nmap OS-Fingerprinting**[5]", Mr. D. B. Berreuta describes various methods for providing misinformation to NMAP. These methods include operating system modules and custom kernel modifications, special protocol stack features, and traffic monitor/response programs. If a rouge operator is intent on hiding a system from NMAP, it is expected that the method described by Mr. Berreuta would be sufficient.

The NESSUS vulnerability assessment tool (http://www.nessus.org/intro.html) is also very useful in identifying rogue nodes on a network. It can provide a tremendous amount of detailed information about hosts that follow standard protocol models. However since it leverages the capabilities of NMAP, it suffers the same problems and cannot reliably detect systems that are specifically configured to avoid detection.

Protocol Analysis

Network protocol analyzers can passively examine network traffic and detect anomalies. The open-source tool ETHEREAL (http://www.ethereal.com) can be used for this, although like most protocol analyzers, it requires an operator with significant skill and experience to correctly identify issues. This approach is most useful for spot checking network traffic flows on low traffic networks. Since it is a manual process, this quickly becomes overwhelming on busy networks.

The commercial SNIFFER™ product from Network Associates (http://www.sniffer.com) has some excellent expert-system analysis tools built in. It can be very useful for examining secure networks for rogue traffic patterns and anomalies, but it is still a labor-intensive process, and is not helpful in detecting passive devices that do not transmit or otherwise announce their presence over the network.

Ethernet Address Monitoring

The Network Research Group at Lawrence Berkeley National Laboratory[6] has made available ARPWATCH, another valuable tool in wide use on highly secure local networks. According to a NASA training document written by Steve Walworth, "It listens for arp

---

[4] Wolfgang, p10

[5] Berreuta, David Barroso. "A Practical approach for defeating Nmap OS-Fingerprinting." 2003. URL: http://voodoo.somoslopeor.com/papers/nmap.html (3 Feb. 2004)

[6] LBL Network Research Group. "Network Tools." Lawrence Berkeley National Laboratory. 2004 URL: http://www-nrg.ee.lbl.gov/nrg.html (4 Feb 2004)

packets on the default ethernet interfaces […] of a host and records changes made to ip/ethernet addresses made by other hosts.[7]"

It works by cataloging and indexing Media Access Control addresses at Layer 2, the Data Link Layer[8]. This protocol layer lies below the transport layer and network layer used by IP in the standard ISO protocol model. In most situations this will be Ethernet, but according to the included documentation, FDDI is also supported[9]. ARPWATCH sends alerts when it notices that a change in addresses assignments. Over time it builds a cross-reference of Ethernet addresses and associated IP addresses. It can detect when IP addresses are stolen, changed, or are simply misconfigured. It is most useful when deployed to alert operators when the protocols underlying IP are being fooled, such as in the case of an ARP POISONING attack.

Unfortunately this tool is not that useful in detecting when the physical Ethernet connection is being tampered with. If an Ethernet repeater or network tap is inserted in the physical path, ARPWATCH remains silent. It is usually configured to alert when the IP address associated with a specific MAC address changes without notice, typically when equipment is replaced or an IP address is reassigned. This may help detect casual network tampering, but it will likely be anticipated by an informed interloper targeting a secure installation, who will take measures to avoid placing traffic on the network that would be detectable by ARPWATCH.

Another Method: Port-Status Monitoring

By actively monitoring port status changes in the Ethernet switches themselves, it is possible to detect and react when a physical connection is being tampered with. On Cisco Catalyst Ethernet switches a port status message is generated whenever an Ethernet node is attached or removed. These messages can then be archived remotely. According to the **Catalyst 3550 Multilayer Switch System Message Guide**, "the system software sends these messages to the console (and, optionally, to a logging server on another system)"[10].

In secure networks, these status messages are usually sent to a logging server for future analysis or debugging purposes. In networks where there are many changes this can result in a large number of messages being generated and sent. According to Dr. Vincent Jones, "in real life we find that this resource is frequently ignored simply because of the difficulty of dealing with the huge quantity of raw data."[11]

---

[7] Walworth, Steve. "ARPWATCH." NASA Network Training Group. 8 Sep 2004.
URL: http://www.nas.nasa.gov/Groups/Networks/Training/ant/arpwatch.html (3 Feb 2004)

[8] Webopedia. "The 7 Layers of the OSI Model." Webopedia.com. 2004
URL: http://www.webopedia.com/quick_ref/OSI_Layers.asp (4 Feb 2004)

[9] LBL Network Research Group, "arpwatch.tar.gz README file." 22 Jan 2004
URL: ftp://ftp.ee.lbl.gov/arpwatch.tar.gz (4 Feb 2004)

[10] Cisco. Catalyst 3550 Multilayer Switch System Message Guide. (San Jose, CA: Cisco Systems Inc. Mar 2003.) p1.1

[11] Jones, p1.

The usefulness of these messages is often greater on static networks with little or no end user systems, such as on secure private networks carrying digital surveillance video traffic or access control system data. Port status monitoring can help filter through large amounts of data and automate the process. Working alongside these other traditional network integrity methods, this method can increase confidence in the security of local network installations.

The processes described in this paper rely upon software source code readily available on the Internet, combined with the author's own scripts.

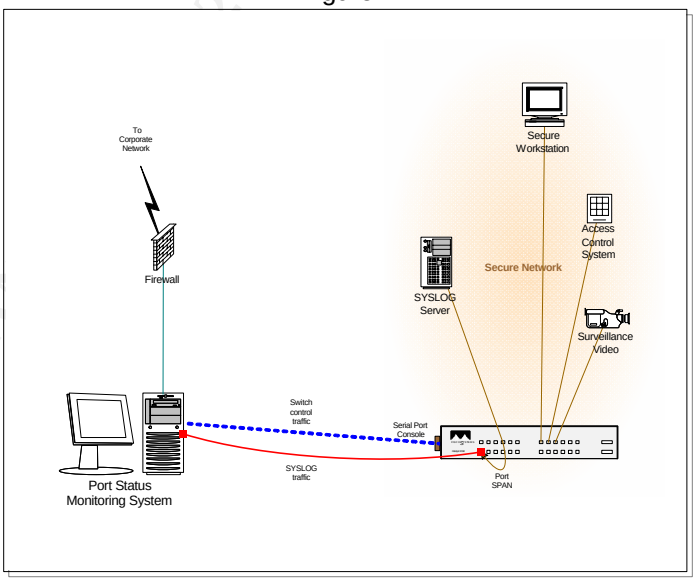
Details of the Port Status Monitoring Method

This method of active port status monitoring can be analyzed by looking at three main parts of the process:

1) The generation of messages at the Ethernet Switch describing interface status changes
2) The transmission of these messages, and their collection by a covert log server
3) Scripts to automate analysis and escalate events

To illustrate this, consider a private local network carrying secure traffic (Please see figure 1).

Figure 1



The components in this scenario are all connected to a Cisco Catalyst 3550 switch, and communicate freely amongst themselves. The network switch is configured to send

SYSLOG messages in Cisco's own format[12] to a local log server. This server would be configured with a standard SYSLOG server, and may also be serving other protocols.

There are many different models in the Catalyst product line, and most of them can be configured to send status messages in a format similar to that which is described here. In a real-world example there would likely be multiple Ethernet switch devices of differing type, however for clarity only one is used in this example.

Other brands of network switches are also able to generate system messages indicating interface state changes, and it is expected that these could be easily integrated into this method. A real-world network should also contain standard IDS (Intrusion Detection System) and ARP (Address Resolution Protocol) monitoring systems, providing comprehensive security coverage.

In this design, traffic sent to the SYSLOG server is copied across a Cisco SPAN (**S**witched **P**ort **AN**alyzer) connection, so that a hidden (or stealth) SYSLOG server can also log the messages. This added security measure effectively separates and isolates the monitoring process. This is especially useful, as a malicious party would not normally be able to detect the monitor, and would not be able to interact with it unless physical access to the device was obtained. The duplication of log data also enhances the confidence in the log file themselves. If log file tampering is suspected, a security analyst can compare the two separate copies and examine them for anomalies.

A custom analysis script runs on the Port Status Monitoring system. This process watches the copied SYSLOG messages as they arrive in the monitoring system, and compares them to a pre-configured list of important interface. If a message is received which refers to one of these interfaces, a specific action is initiated, based upon a severity rating previously chosen for that interface. A list of these interfaces, and their associated severity levels, is contained in a configuration file. The actions performed upon escalation can range from simple e-mail and pager alerts to emergency remote shutdowns. These actions are defined in the process script running on the monitoring system.

The design of the script allows for customization of the number of severity escalation states available, and the specific actions desired for each severity level. For example, it is possible to configure the script to send special configuration commands to the switch itself in order to force an administrative block on a particular interface, or even to send a coded message to many hosts initiating a system-wide security emergency shutdown. (It should be noted that some of these actions are suitable for use in only the most tightly controlled secure environments.)

Testing Environment

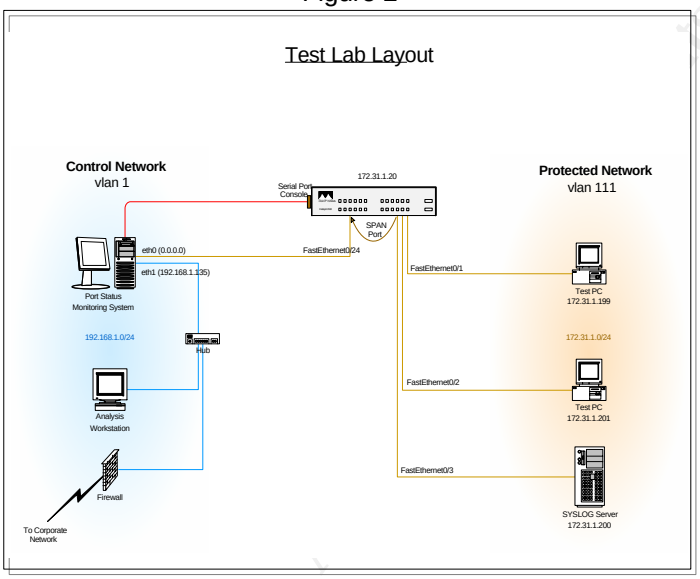
The majority of the hands-on research for this paper was performed in the author's test lab, using a variety of common PCs running Windows XP, Red Hat Linux 8.0, and

[12] Cisco. Catalyst 3550 Multilayer Switch Software Configuration Guide. p15.2

Knoppix 3.3. Knoppix is a version of Linux that is configured to run completely from a single boot CD. It is available at http://www.knopper.net/knoppix/index-en.html.

The primary Ethernet switch used to generate port status messages was a Cisco Catalyst 3550 running IOS 12.1(13)EA1a. The network environment used during the research is illustrated in the following diagram:

Figure 2



The protected network is defined as VLAN111 on the switch, and assigned the IP subnet of 172.31.1.0. This private network is not routed.

The port monitoring system is attached to the interface FastEthernet0/24. This interface is configured in 'monitor' mode to span only the interface FastEthernet0/3, which is connected to the SYSLOG server. All traffic sent to and from the SYSLOG server is duplicated here, and visible to the port monitoring system via this SPAN port.

A non-usable IP address of 0.0.0.0 is assigned to the Ethernet interface on the monitoring system that is connected to the SPAN port. An IP address is usually required to enable an interface; however the actual address used is not important here, as no traffic is expected to be sent out of this interface and back into the protected network. (Additionally, a local firewall process can be used to further protect and block any accidental outbound traffic from leaving on that interface.)

Choosing 0.0.0.0 instead of a normal address has another benefit. It makes it more difficult for an interloper to know that raw traffic is being watched. According to "Sniffin' the Ether v2.0" by Aleric, it is advisable to "…configure the interface with the address of 0.0.0.0. This will allow the sniffer to monitor traffic but to not be detected.[13]"

[13] Aleric. "Sniffin' the Ether v2.0" Unixgeeks.org. 2002
URL: http://www.unixgeeks.org/security/newbie/security/sniffer/sniffer.html#hard (4 Feb 2004) Section IX

For the system to function correctly, it is necessary for the port monitoring system to have a second network interface for the transfer of alert messages to the corporate network.

For added safety this may be done by using a generic NAT (Network Address Translation) firewall for connection to the rest of the corporate network. No inbound connections are allowed in through this firewall, but outbound e-mails and other types of alert messages, such as paging and other messaging protocols as defined in the PARSER.PL script are allowed to be sent out. This design provides the monitoring system with another layer of protection from attack via the corporate network.

Testing Procedures

The Cisco Catalyst 3550 used in the test lab was configured in the following manner:

```
!
! Last configuration change at 10:30:59 PST Tue Jan 27 2004 by cisco
! NVRAM config last updated at 16:18:19 PST Tue Jan 27 2004 by cisco
!
version 12.1
no service pad
service timestamps debug datetime msec localtime show-timezone
service timestamps log datetime msec localtime show-timezone
!
hostname 3550PWR-TEST
!
no logging buffered
logging rate-limit 300
aaa new-model
aaa authentication login default local
aaa authorization exec default local
!
clock timezone PST -8
clock summer-time PDT recurring
ip subnet-zero
!
spanning-tree mode pvst
spanning-tree extend system-id
!
interface FastEthernet0/1
 switchport access vlan 111
 switchport mode access
 no ip address
!
interface FastEthernet0/2
 switchport access vlan 111
 switchport mode access
 no ip address
!
interface FastEthernet0/5
 switchport access vlan 111
 switchport mode access
 no ip address
!
interface FastEthernet0/7
 switchport access vlan 111
 switchport mode access
 no ip address
!
interface FastEthernet0/19
switchport access vlan 111
 switchport mode access
 no ip address
!
interface FastEthernet0/24
 description SPAN Monitor for port 3
 switchport mode access
 no ip address
!
interface Vlan111
 ip address 172.31.1.20 255.255.255.0
```

```
!
ip default-gateway 192.168.1.1
ip classless
no ip http server
!
!
logging 172.31.1.200
!
line con 0
line vty 5 15
!
ntp clock-period 17180390
ntp server 192.168.1.1
!
monitor session 1 source interface Fa0/3
monitor session 1 destination interface Fa0/24
end
```

Note that for clarity, several unused interface configuration lines have been omitted from this listing. The configuration lines of particular interest in this discussion are highlighted in bold face above, and are listed here:

logging 172.31.1.200

monitor session 1 source interface Fa0/3

monitor session 1 destination interface Fa0/24

The logging command is necessary in order to tell the switch to send SYSLOG messages over the network to a local logging server. These messages will be collected and saved by whichever SYSLOG process is running there. The monitor commands are needed so that this traffic will also be copied from Fa0/3, and out to the port monitoring system on Fa0/24, where they are covertly logged and analyzed by the port state monitoring system. The syntax for these configuration commands is available in chapters 24 and 26 of Cisco's Catalyst 3550 Multilayer Switch Software Configuration Guide.

The port status monitoring system described here can run on top of Knoppix 3.3, Red Hat Linux 8.0, or any other similar operating system. Some components depend upon the PERL scripting language (http://www.perl.com), and the LIBPCAP programming library (http://sourceforge.net/projects/libpcap/), both of which must be installed on the system. They are included in the default Knoppix 3.3 distribution.

The monitoring system is comprised of two parts. The first part is a customized SYSLOG daemon called PSYSLOGD[14]. This program is written by Nathan Bates and relies upon the LIBPCAP programming library to get access to raw Ethernet data transmitted across the network. This method of listening directly to the data sent over the Ethernet cable is called 'promiscuous mode'. The LIBPCAP library is a common interface used by programmers to access this mode and gain visibility to the raw data.

Once compiled, the PSYSLOGD program can be configured to listen on the network specifically for SYSLOG message traffic, and then extract those messages and save

[14] Bates, Nathan.. " [logs] a promiscuous syslog daemon". Securityfocus.com. Jan 30, 2002 URL: http://www.securityfocus.com/archive/116/253092/2002-01-27/2002-02-02/0 (3 Feb 2004)

them to a local log file in normal SYSLOG format[15]. It can coexist easily with other SYSLOG process so long as they are configured to log only messages generated by the operating system, leaving the network messages to be handled exclusively by PSYSLOGD. This is the normal default configuration state for both Knoppix and Red Hat Linux.

During lab testing, PSYSLOG was compiled from the source code (as published on SecurityFocus.com) and invoked by the 'root' account in the following manner:

```
ifconfig eth0 0.0.0.0 promisc up
./psyslogd -i eth0 udp port syslog and broadcast >> /var/log/syslog &
```

These commands configure the first Ethernet interface eth0 as the listening port with a dummy IP address, and then start the promiscuous daemon, listening to all SYSLOG messages sent over the network, regardless of their intended destination. Output from PSYSLOGD is in standard SYSLOG format, and is shown here being appended to the main system SYSLOG file, simulating the output of a normal SYSLOG process.

The second part of the monitoring system is PARSER.PL, a custom script written in PERL that independently examines log file entries and performs predefined actions based upon keyword matches. This script reads a configuration file "tds.conf" for advice on the specific keywords to watch and actions to perform. The sample configuration file contains the following:

```
# IPADDR,InterfaceX/Y,normalstate,severity
#---------------,----------------,----,-
172.31.1.20,Vlan111,up,1
172.31.1.20,FastEthernet0/1,up,1
172.31.1.20,FastEthernet0/2,down,2
172.31.1.20,FastEthernet0/3,up,3
172.31.1.20,FastEthernet0/4,down,2
172.31.1.20,FastEthernet0/23,down,1
172.31.1.20,FastEthernet0/24,monitor,1
```

The system administrator populates this configuration file with data pertaining to the local network installation. The data is delimited by the comma character. The first two fields refer to the IP address of the Ethernet switch making the report, and the particular interface being scrutinized. The third field records the normal operating state of this interface. The last field is the desired alert level; used to invoke a specific class of action should the state of the interface change from normal.

The requirements here are much simpler than those described by Dr. Jones[16], as we only need to track physical layer events to achieve our tamper detection goals. Therefore the PARSER.PL script provided here by the author uses much simpler logic. Additionally the author's script examines log entries in real time as they are written, and automatically launches predefined actions.

---

[15] Lonvick, C. "RFC 3164 - The BSD Syslog Protocol." FAQS.ORG Internet RFC/STD/FYI/BCP Archives. Aug 2001 URL: http://www.faqs.org/rfcs/rfc3164.html (4 Feb 2004)

[16] Jones, pp 5-8

Testing Results

During testing the PSYSLOGD program was started and the PARSER.PL was invoked. The script waited for SYSLOG messages of a particular type to be written to the log file. Messages not matching the Cisco format were ignored, as the script scans for messages containing only **"%LINEPROTO-5-UPDOWN**" references. These are the messages important for tracking the state of Ethernet connections in the switch. When a message of this type was received, the script tried to match the originating IP address and interface description with an entry from the configuration file. If there was a match, the "up" or "down" state was compared with the normal status indicated in the configuration file. If it was found to be different, the appropriate escalation procedure was initiated. The process then continued to listen for further messages.

If a message was recognized as notice of an interface returning to normal state, a log message was printed, but no further escalation was performed, as the script considers all events of this type to have an alert level of zero.

When an Ethernet patch cord was disconnected from a PC while the PSYSLOGD program and the PARSER.PL script were running, status messages were seen to be generated and transmitted to the monitoring system, where the script correctly performed the predefined actions.

The results of a sample monitoring session are listed in the table below:

```
ALERT1: Jan 28 14:10:33.894066  172.31.1.20     FastEthernet0/1 Alert Level 1 - sending e-mail
NOTICE: Jan 28 14:10:35.896546  172.31.1.20     FastEthernet0/1 Back to normal state.
ALERT1: Jan 28 14:10:44.973432  172.31.1.20     FastEthernet0/1 Alert Level 1 - sending e-mail
NOTICE: Jan 28 14:10:46.975865  172.31.1.20     FastEthernet0/1 Back to normal state.
ALERT1: Jan 28 14:10:56.891354  172.31.1.20     FastEthernet0/1 Alert Level 1 - sending e-mail
ALERT2: Jan 28 14:11:01.263671  172.31.1.20     FastEthernet0/2 Alert Level 2 - initiating incident!
NOTICE: Jan 28 14:11:08.300204  172.31.1.20     FastEthernet0/2 Back to normal state.
NOTICE: Jan 28 14:11:10.783183  172.31.1.20     FastEthernet0/1 Back to normal state.
ALERT3: Jan 28 14:11:20.782136  172.31.1.20     FastEthernet0/5 Alert Level 3 - PAGING SECURITY OFFICER!
page sent to 6045551212
NOTICE: Jan 28 14:11:57.697775  172.31.1.20     FastEthernet0/5 Back to normal state.
ALERT1: Jan 28 14:12:20.336526  172.31.1.20     FastEthernet0/19        Alert Level 1 - sending e-mail
ALERT1: Jan 28 14:12:20.336798  172.31.1.20     Vlan1   Alert Level 1 - sending e-mail
NOTICE: Jan 28 14:12:22.338995  172.31.1.20     FastEthernet0/19        Back to normal state.
NOTICE: Jan 28 14:12:52.17709   172.31.1.20     Vlan1   Back to normal state.
ALERT1: Jan 28 14:13:20.933443  172.31.1.20     FastEthernet0/19        Alert Level 1 - sending e-mail
ALERT1: Jan 28 14:13:20.933716  172.31.1.20     Vlan1   Alert Level 1 - sending e-mail
ALERT2: Jan 28 14:13:28.616158  172.31.1.20     FastEthernet0/7 Alert Level 2 - initiating incident!
NOTICE: Jan 28 14:14:08.891004  172.31.1.20     FastEthernet0/7 Back to normal state.
NOTICE: Jan 28 14:14:12.973937  172.31.1.20     FastEthernet0/19        Back to normal state.
NOTICE: Jan 28 14:14:41.972097  172.31.1.20     Vlan1   Back to normal state.
```

As shown here, the script appears to correctly recognize important state changes and correctly initiates actions as defined by the system administrator. At "Alert Level 1" an e-mail is sent to the local system account. At "Alert Level 2" an e-mail is sent to the site administrator for follow-up and action. When an event is noticed that is associated with "Alert Level 3", the script will page a security operator and also send an e-mail to the site administrator. This result is seen in the above example on interface FastEthernet0/5.

The content of the e-mail messages can be different for each alert level. The messages themselves can be changed by editing the PARSER.PL script. If desired, additional alert levels can also be added here as well.

Please refer to the appendix for listings of the source code for the PARSER.PL script.

Conclusions

The processes described here provide a useful method for actively monitoring Ethernet port status. This is accomplished by covertly collecting SYSLOG entries, filtering for Cisco Catalyst port status messages, and then running a script to initiate any of several pre-defined escalation actions. This fills a need in some highly secure private networks to know quickly when certain devices are physically removed from the network. This method can increase confidence in the integrity of secure private networks, especially when the system is covertly configured to not be easily detectable on the protected network.

Unfortunately this method may not yet satisfy all requirements that operators of highly secure networks might have. At this time the script has no way to detect or escalate action when the SPAN port is disconnected, or even to detect when the switch is powered off. This may yet be addressed by adding a watchdog process that independently monitors the state of the SPAN monitor and perhaps also the basic functionality of the switch. Any interruption of these should raise a high-severity alert.

Another improvement would be to enhance the standard SYSLOG functions with versions that support secure or authenticated transport. Some of these enhancements are available[17], but better tools and greater acceptance by network device vendors are anticipated for the future. This would be an excellent way to ensure message integrity, and would remove the possibility that the monitoring system could be triggered by false messages maliciously injected into the network.

In spite of these deficiencies, this method of active port-status monitoring for tamper detection should be considered as a useful security enhancement for secure private networks.

[17] Mietus, Albert. “Secure BSD syslog” Sourceforge.net 27 Mar 3002 URL: http://sourceforge.net/projects/syslog-sec/ (4 Feb 2004)

Appendices

PSYSLOGD.C

The source code listing for PSYSLOGD.C appears in an article entitled "[logs] a promiscuous syslog daemon", posted by Nathan Bates to the SecurityFocus.com "loganalysis" mailing list on Jan 30, 2002. It can be obtained from Securityfocus.com at http://www.securityfocus.com/archive/116/253092/2002-01-27/2002-02-02/0

PARSER.PL

This script requires the PERL module "File::Tail" to be installed on the monitoring system. If your Linux distribution does not include this module by default, it can be easily downloaded from http://search.cpan.org/~mgrabnar/File-Tail-0.98/Tail.pm, or installed by using the PERL CPAN utility at http://search.cpan.org/~jhi/perl-5.8.0/lib/CPAN.pm.

```
#!/usr/bin/perl

use strict;
use File::Tail;


my $configfile = "./tds.conf";
my $logfile = '/var/log/syslog';   # or perhaps /var/log/messages, or wherever psyslogd puts it
my @data;
my $count = 0;
# email addresses to alert at various levels
my @email;
$email[1] = 'root@localhost';
$email[2] = 'security@example.com';
$email[3] = 'security@example.com';
my $pagealert = "./page-me.sh 6045551212";




sub alert {
  #Perform an action based on what data we found in the log.
  # the keyword argument contains the escalation level
  # if keyword is '0', it is assumed that the status has reverted to the value in the config file

  my ($host, $interface, $timestamp, $keyword, $line) = @_;

  if ($keyword == 0)
        {
          printf STDERR "NOTICE: %s\t%s\t%s\tBack to normal state.\n", $timestamp, $host, $interface ;
        }
  elsif ($keyword == 1)
        {
          printf STDERR "ALERT1: %s\t%s\t%s\tAlert Level 1 - sending e-mail\n", $timestamp, $host,
$interface;
          my $msg = "|mail -s ALERT1 $email[1]";
          open MAIL, $msg or die "cannot pipe to mail: $!";
          printf MAIL "ALERT: Level 1\n";
          printf MAIL " Host: %s\n", $host;
          printf MAIL " Interface: %s\n", $interface;
```

```
          printf MAIL " Time: %s\n", $timestamp;
          printf MAIL "\n%s\n", $line;
          close MAIL;
        }
  elsif ($keyword == 2)
        {
          printf STDERR "ALERT2: %s\t%s\t%s\tAlert Level 2 - initiating incident!\n", $timestamp,
$host, $interface ;
          my $msg = "|mail -s ALERT2 $email[2]";
          open MAIL, $msg or die "cannot pipe to mail: $!";
          printf MAIL "ALERT: Level 2\n";
          printf MAIL " Host: %s\n", $host;
          printf MAIL " Interface: %s\n", $interface;
          printf MAIL " Time: %s\n", $timestamp;
          printf MAIL "\n%s\n", $line;
          close MAIL;
        }
  elsif ($keyword == 3)
        {
          printf STDERR "ALERT3: %s\t%s\t%s\tAlert Level 3 - PAGING SECURITY OFFICER!\n", $timestamp,
$host, $interface ;
          system "/bin/sh", "-c", $pagealert;
          my $msg = "|mail -s ALERT3 $email[3]";
          open MAIL, $msg or die "cannot pipe to mail: $!";
          printf MAIL "ALERT: Level 3\n";
          printf MAIL " Host: \t%s\n", $host;
          printf MAIL " Interface: \t%s\n", $interface;
          printf MAIL " Time: \t%s\n", $timestamp;
          printf MAIL " Message: \t%s\n\n\n", $line;
   printf MAIL " ****************************************\n";
   printf MAIL " * NOTE: This is a level 3 alert!            *\n";
   printf MAIL " *       A Security Officer has been paged!  *\n";
   printf MAIL " ****************************************\n\n";
          close MAIL;
        }
}


sub read_conf {
  open FILE, $configfile
    or die "Can't open '$configfile': $!";

  while (<FILE>) {
    chomp;
    if ( (index($_,"#")) && (length) ) {  #look for comments and blank lines

      my @line = split /,/;
      if ( (@line == 4) ) {
        for my $loop (0..3) {
          $data[$count][$loop] = @line[$loop];
          #printf "debug: %i %i\n", $count, $loop;
        }
        #printf "debug: %s\t%s\t%s\t%s\n", $data[$count][0], $data[$count][1], $data[$count][2],
$data[$count][3];
        $count++; # good line, so count it
      } else {
        #printf "debug: ---bad data---\n";
      } #endif data

    } else {
      #printf "debug: ---comment---\n";
    } #endif line

  } #end file

  printf STDERR "Read %i items from %s\n", $count, $configfile;
  close FILE;
}

###################################
# Main


# read the configuration file to find out which interfaces we're watching
&read_conf;

# tail the log file and parse messages
my $ref=tie *FH,"File::Tail",(name=>$logfile, maxinterval=>3, adjustafter=>2);
printf STDERR "Watching %s\n\n", $logfile;
while (<FH>) {
```

```
   chomp;
   #printf "debug: %s\n", $_;

   #check to see if this line contains a suitable Cisco port state message
   if (index($_, " %LINEPROTO-5-UPDOWN: ") == -1) { next; }

   # check if hostname is present
   my $loop = 0;
   until ($loop==$count) {

     #printf "debug: looking for host %s\n", $data[$loop][0];
     if ( index($_, " ".$data[$loop][0]." ") >= 0 ) {   #found hostname
       #printf "debug: found host $data[$loop][0]\n";
       #printf "debug: looking for interface %s\n", $data[$loop][1];
       if ( index($_, " ".$data[$loop][1].", ") >= 0 ) {   #found interface
         #printf "debug: found interface %s (%i)\n", $data[$loop][1], index($_, $data[$loop][1]);
         #matches! Now do stuff...
         my $host = $data[$loop][0];
         my $iface = $data[$loop][1];
         my $tmstmp = substr($_,0,22);
         my $cword = substr($_, rindex($_," "));
         #decision logic
         if ( substr($_, rindex($_," ")) =~ $data[$loop][2]) {
           #now in normal state
           &alert($host, $iface, $tmstmp, 0, $_);
         } else {
           #problem!
           &alert($host, $iface, $tmstmp, $data[$loop][3], $_);
         }
         last;
       } #endif interface
     } #endif host

     $loop++;

   }
}
```

References

Aleric. “Sniffin’ the Ether v2.0” Unixgeeks.org. 2002
URL: http://www.unixgeeks.org/security/newbie/security/sniffer/sniffer.html#hard
(4 Feb 2004) sect. IX

Bates, Nathan. " [logs] a promiscuous syslog daemon". Securityfocus.com. Jan 30, 2002
URL: http://www.securityfocus.com/archive/116/253092/2002-01-27/2002-02-02/0
(3 Feb 2004)

Berreuta, David Barroso. “A Practical approach for defeating Nmap OS-Fingerprinting.”
2003. URL: http://voodoo.somoslopeor.com/papers/nmap.html (3 Feb. 2004)

Cisco. Catalyst 3550 Multilayer Switch Software Configuration Guide.
(San Jose, CA: Cisco Systems Inc. Dec 2002.) ch.15, 26

Cisco. Catalyst 3550 Multilayer Switch System Message Guide.
(San Jose, CA: Cisco Systems Inc. Mar 2003.) ch.1

Jones, Dr. Vincent C. “Automated Analysis of Cisco Log Files.”Networkingumlimited.com.
1999 URL: http://www.networkingunlimited.com/white007.html (4 Feb 2004)

Lonvick, C. “RFC 3164 - The BSD Syslog Protocol.”
FAQS.ORG Internet RFC/STD/FYI/BCP Archives. Aug 2001
URL: http://www.faqs.org/rfcs/rfc3164.html (4 Feb 2004)

LBL Network Research Group. “Network Tools.” Lawrence Berkeley National Laboratory.
2004 URL: http://www-nrg.ee.lbl.gov/nrg.html (4 Feb 2004)

Mietus, Albert. “Secure BSD syslog” Sourceforge.net 27 Mar 3002
URL: http://sourceforge.net/projects/syslog-sec/ (4 Feb 2004)

Schwartz, Randal L., Phoenix, Tom. Learning Perl Third Edition. (Sebastabol, CA.
O’Reilly & Associates, Inc. July 2001.)

Walworth, Steve. “ARPWATCH.” NASA Network Training Group. 8 Sep 2004.
URL: http://www.nas.nasa.gov/Groups/Networks/Training/ant/arpwatch.html
(3 Feb 2004)

Webopedia. “The 7 Layers of the OSI Model.” Webopedia.com. 2004
URL: http://www.webopedia.com/quick_ref/OSI_Layers.asp (4 Feb 2004)

Wolfgang, Mark. “Host Discovery with nmap”, Moonpie.org. Nov. 2002.
URL: http://moonpie.org/writings/discovery.pdf (3 Feb. 2004)