

# Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

This article concludes the "Defending The Homeland" series on intrusion detection. It is the companion to SACLs: Filtering suggestions and ideas. This month we'll review smurf and fraggle attacks and the static access control list (SACL) filtering options you can use to defend your IP network. As an added bonus, a script for dynamically updating your network's ingress SACLs from potential smurf amplifiers is provided.

## What is a smurf/fraggle attack?

The smurf and fraggle attacks are denial of service (DoS) attacks that exploit an IP prefix network and broadcast address. The smurf attack (named after the exploit program written by Tfreak in 1997) uses Internet Control Message Protocol (ICMP) echo reply packets to disable the target system. The fraggle attack (a variation of the smurf code, also written by Tfreak) uses User Datagram Protocol (UDP) echo packets directed at the Unix UDP services echo (port 7), chargen (port 19), daytime (port 13) and qotd (port 17).

## Reconnaissance

These exploits, like most others, are implemented in a two-phased approach. The first phase is resonance. The attacker first collects "trigger" or "amplifier" networks. These are needed to execute the attack. To qualify as a potential amplifier, the network (specifically the router) simply needs to have IP broadcast forwarding enabled. Since this was the default state of most routers and workstations when this attack was devised, smurf amplifier networks were easy to find.

Typical smurf amplifier candidates include large co-location facilities or Web farms. But basically any large concentration of hosts sharing a common network prefix with reasonable Internet access bandwidth can be used. Taking the proper precautions to prevent your network from being used as an amplifier is crucial. The seriousness of an attack depends upon the number of amplifiers (and number of hosts on each) used. With just a few amplifier networks of relative size and a modem (56k) connection, an attacker can launch a sizable attack against a target host. Attackers commonly scan for amplifier networks using tools like Broadscan. There are also black- and white-hat Web sites that list routable prefixes that support IP broadcasts.

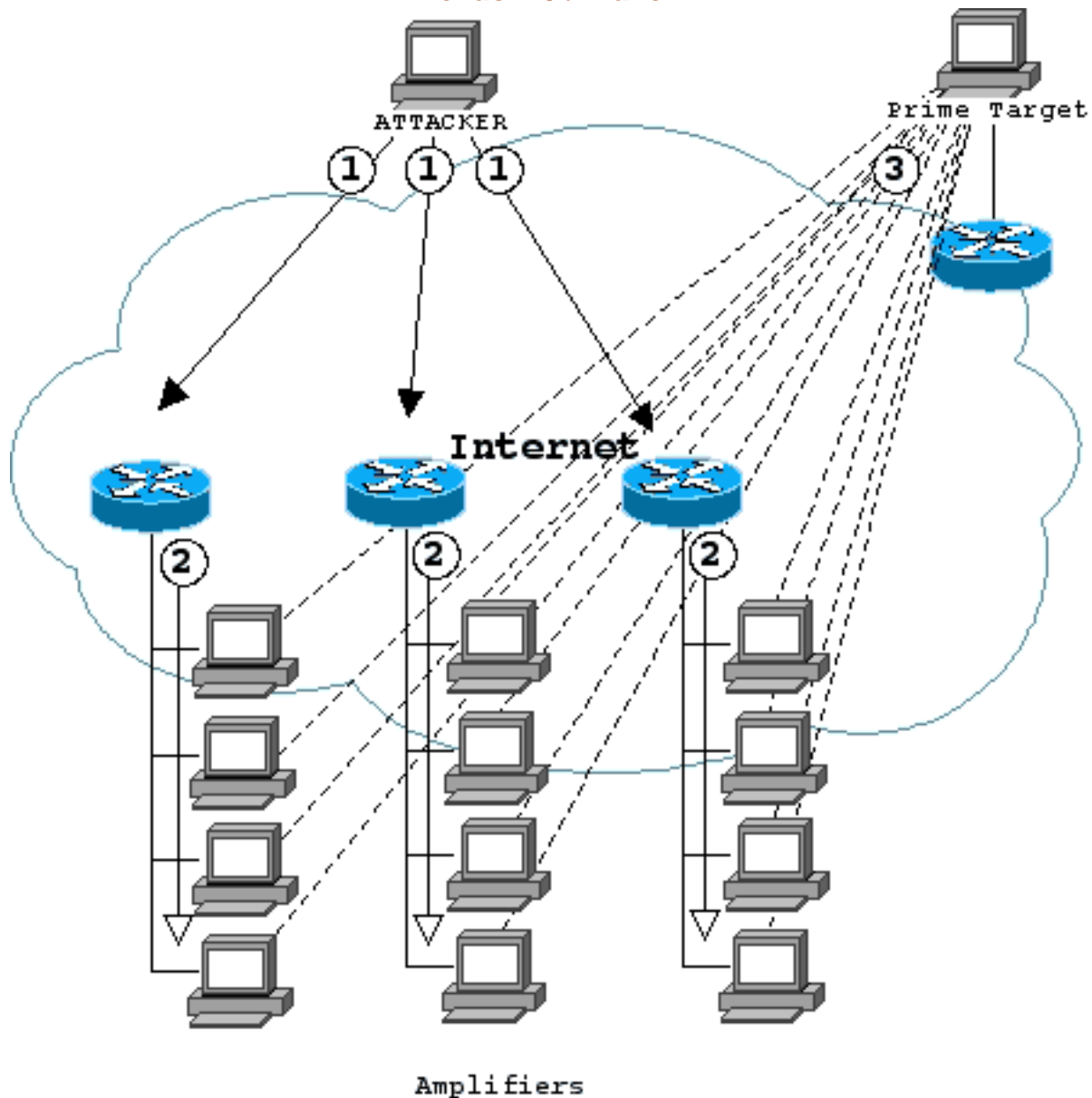
Once the attacker has harvested amplifiers, the next step is to "recon" the victim. This involves first identifying potential targets by looking for critical hosts using DNS, MX records, etc. Once the targets have been discovered, the next step is to identify the types of attacks that can be executed by scanning the system with host scanner such as nmap or netdiag.

## The Attack

The second phase is the attack. What makes smurf attacks unique is the dual nature of the victim. A smurf attack involves three elements: the attacker, the amplifier networks, and the primary target. While the degree varies, both the amplifier and the primary target suffer the effects of the attack.

## Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin



The attacker sends (1) 1500-byte (the Ethernet maximum MTU size) ICMP echo messages to the broadcast address of the amplifier networks. The ICMP messages have the spoofed IP address of the prime target as their source address. The gateway router of the prefix receiving (2) the broadcast ping, forwards the ICMP messages to the adjacent hosts. The host's reply (3) with ICMP echo-reply messages is sent to the primary target's IP address. The target host is inundated with ICMP messages. Forced to deal with the excessive ICMP load, the server is unable to respond to genuine service requests.

# Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

With the fraggle variation, the attack anatomy is the same, except UDP packets are sent instead of ICMP messages. While not as lethal as smurf, there is the added potential for the attacker to implement a ping-pong attack. The attack is initially launched at the chargen service port (19) with the spoofed IP source address of the target and a source service port of 7. This effectively creates a loop, with the character generator service sending out traffic to echo service, which sends the traffic back to the character generator (an attack common to NT systems, often resulting in a system crash) resulting in even more traffic, then the attack load.

Regardless of the attack method, these attacks are quite effective at generating an enormous amount of network traffic. This often results not only in the disabling of the target system, but decreased overall performance of the targets network and the networks (often unknowingly) functioning as amplifiers. That makes this a very lethal form of DoS attack. The number of smurf and fraggle attacks over the Internet has dropped significantly since the attack was first released. This is largely attributed to network administrators taking the appropriate measures to protect their networks and hosts from being vulnerable to the exploit. This attack is still quite viable within private networks where the proper defenses are not implemented due to the often "trusting" attitude toward employees and company assets. The attitude persists, despite numerous surveys finding that 70% of security incidences originate internally. Businesses that implement new public networks and often work from the "security thru obscurity" assumption, "I am new, no one will notice me." The fact is that most of these "pre-made attacks," are run by script kiddies and crackers who are attacking for the thrill, not for any real reason, and are always looking for new prey.

## Smurf/fraggle Attack Detection

The most common tip-off that you are either under attack or functioning as an amplifier is sluggish network or server performance. Other indicators are:

- High router CPU utilization.
- Asymmetric traffic patterns. Targets will see excessive inbound and minimal outbound traffic. Amplifiers will see excessive outbound with average inbound traffic levels
- Unexplained system crashes.
- IP traffic from multiple hosts with the same IP prefix.
- IP traffic with destination addresses ending with .255 or .0

Evidence of one or more of these operational anomalies warrants further investigation.

## Smurf/fraggle SACL Filtering Options

For starters, disabling IP broadcast capabilities is a must. If you are running IOS 11.x or earlier, the interface command <no ip broadcast > should be applied to each router interface. If you're running IOS 12.x or later, this is enabled by default. This action eliminates the potential of your network being used as an amplifier. The best defense against smurf and fraggle attacks is to filter the attack points or disable the exploitable services. Last month's article, SACLs: Filtering suggestions and ideas, covered options for filtering ICMP and other TCP/UDP services.

## Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

Of the two exploits, the fraggle attack is far easier to mitigate. None of the services are vital for operation on Unix or NT systems. Most Unix implementations ship today with the UDP and TCP small servers (echo, chargen, daytime, qotd, and discard) disabled. NT requires the installation of Simple TCP/IP Services. However, Cygwin users should beware that all of the small servers and most of the standard network service daemons are enabled by default, requiring some editing of the /etc/inetd.conf file before attaching the system to a public (or really any) network.

IOS versions 11.x and earlier have them enabled by default, and IOS versions 12.x and later (after discovering a buffer overflow vulnerability) have them disabled by default. The IOS commands <(no)servers tcp-small-servers> and <(no)servers udp-small-servers> can be used to enable or disable the services. They exist on all three platforms solely for diagnostic purposes. So, depending on your environment, you may have little if any default exposure. Common best practice, however, is to filter these services (at a minimum) inbound on the Internet gateway's public facing interface. Here is the SACL filtering syntax:

```
access-list 100 deny tcp any any eq 7 log ! Echo Service
access-list 100 deny udp any any eq 7 log! Echo Service
access-list 100 deny tcp any any eq 9 log ! Discard Service
access-list 100 deny tcp any any eq 19 log! Character Generator chargen
access-list 100 deny udp any any eq 19 log! Character Generator chargen
access-list 100 deny tcp any any eq 13 log! daytime service
access-list 100 deny udp any any eq 13 log! daytime service
access-list 100 deny tcp any any eq 17 log! qotd service
access-list 100 deny udp any any eq 17 log! qotd service
```

To defend against smurf attacks, a few approaches can be taken. The first is to simply block inbound and outbound ICMP echo and ICMP echo-reply packets. This option is rather harsh, because it completely disables the option to use the TCP/IP diagnostic tool. But it protects hosts from both originating and being the target of a smurf attack, eliminates recon pings and OS signature scans. You need to decide if the trade-offs are worth it. Here is the SACL filter syntax:

```
access-list 100 deny icmp any any echo log-input ! Disables remote ping
access-list 100 deny icmp any any echo-reply log-input ! Disables local ping
```

A more accommodating alternative is to block inbound ICMP echo-reply packets on a host-by-host basis. Here is an example:

```
access-list 100 deny icmp any host 192.168.100.30 echo-reply log-input
access-list 100 deny icmp any host 192.168.100.32 echo-reply log-input
access-list 100 deny icmp any host 192.168.100.45 echo-reply log-input
access-list 100 deny icmp any host 192.168.100.90 echo-reply log-input
```

This option leaves the ability to use ping intact (for hosts that are not filtered) but defends those that need it. The downside is that it requires more administrative work, because the SACL needs to be updated whenever there are host address changes (leaving open the chance for something to be missed).

## Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

A third option is to leave ICMP unfiltered but use committed access rate (CAR) traffic filtering. This option is appealing because it also defends against ping of death attacks (i.e., Jolt, SSPING, nestea.c, snork.c, lan.c, killwin.c and Winnuke) and some distributed denial of service attacks (Trinno and stacheldrad) that utilize ICMP to implement the attack and provide client-> handler -> agent communication. Cisco provides CAR configuration examples and usage guidelines in a Distributed Denial of Service Newsflash. CAR functionality is applied to a specific interface, using a reference ACL for traffic qualification. The interface configuration command is <rate-limit <input/output> access-group <ALC#> <total Bandwidth in BPS> <Normal Burst in Bytes % / [8 / TB-BPS]> <Max Burst in Bytes % / [8 / TB-BPS]> conform-action <transmit|drop> exceed-action transmit|drop>. Here is an example CAR statement for a T1 serial link, allowing up to 128K of ICMP echo-reply traffic:

```
interface Serial 0/0
ip address 172.30.1.0 255.255.255.0
rate-limit input access-group 101 1544000 64000 128000 conform-action
transmit exceed-action drop
!
access-list 100 permit icmp any any echo-reply
```

The last approach is geared toward environments where port/protocol-based filtering presents a problem. This tactic consists of simply filtering incoming traffic from networks verified to be potential smurf amplifiers.

### Dynamic Smurf Amplifier Filtering

Crackers and script kiddies are not the only ones scanning the Internet for IP broadcast-enabled networks. Two sites <http://www.powertech.no/smurf/> and <http://www.netscan.org/> scan the Internet for networks that support directed broadcast forwarding. Both post lists of potential amplifier networks. These "white hat" efforts allow network administrators to both check their networks to ensure that they are not potential amplifiers and to filter traffic from confirmed amplifier networks. Since the status of these networks can change on a regular basis, dynamic updating of router SACLs is an effective way to ensure the protection of your network from publicly originated smurf attacks.

A few months back, I posted an article on Automating router configurations and backups with Expect. The article reviewed configuring TFTP services on the Unix platform and provides some generic Expect scripts for backing up and uploading router configurations. Based on that previous work, here is a scripting example that downloads Powertech's smurf amplifier list using GNU's wget application and reformats it into an IOS IP named access list (the only ACL format that permits editing). It then retracts the previous SACL entries and adds the new entries.

The scripts run on any Unix or Cygwin-equipped NT system with the borne or bash shell, Expect and wget. To upload and download SACL configurations, a TFTP server is needed. The SACL creation script is called smurf-filter.sh and the upload/download script is called config-eng.exp. The TFTP server should reside locally on the system used to execute the scripts. See the Automating router configurations article for instructions on setting up TFTP. To make things easier, the script calls a copy of the config-eng.exp script called acl-edit.exp to upload the SACL

# Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

edits and acl-edit-rm.exp to retract SACL edits. All of the script files should be installed in the same directory.

```
[trinity:/usr/local/smurf-UD] mmartin% ls
README acl-edit.exp local-additions smurf-filter.sh
acl-edit-rm.exp archive igw-wrapper.cron
[trinity:/usr/local/smurf-UD] mmartin%
```

Here is the /bin/ls output of the directory where scripts have been installed. Once you have downloaded the script files, you may need to set the execution bit using the command /bin/chmod <script name> ugo+x. When the script runs, a named IP extended inbound SACL is assumed to be installed. This SACL should provide the basic RFC recommended filtering precautions (a sample is provided) except for IP spoofing filtering, which is provided by the tool.

The named SACL is applied to the router's Internet gateway interface using the interface configuration command <ip access-group xxxxx in>.

When running the script, uploading and downloading of the SACL files should be done from behind the filtered interface so modifications to the inbound SACL will not affect communications with routers. Script execution requires three command line interface (CLI) variables: first, the Internet GW public interface address; second, the local IP prefix; and third, the local IP prefixes mask in Cisco's wildcard (reverse-mask) format. Here is a CLI example:

```
root@trinity# ./smurf-filter.sh 172.56.100.2 172.30.1.0 0.0.0.255
```

Now, let's take a look at what the smurf-filter.sh script does and what you need to edit to get it up and running. The script functions are documented at each function. I have added additional comments in some places to provide tips or further explanation. The first section defines variables:

```
#!/bin/sh
#
# Administrator defined variables
#
# Define where the script files are located
APD="/usr/local/smurf-UD"
# Define the name of your router's Internet interface inbound SACL
aclname="ip access-list extended internet-inbound"
# Define the inside address of your Internet router
RTR="172.30.71.1"
# Define the local directory for your TFTP server
TFTP="/tftpboot"
# Define the list of users that you want to receive the comparison
# report of the previous and current SACL update
RPTUSER="mj0u812@yahoo.com"
# Define the local directory where you want the SACL update archives to
# be stored. It should be a subdirectory within the directory where you
# have this script and the expect loader scripts.
```

# Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

```
ARCHDIR="/usr/local/smurf-UD/archive"  
#
```

The script operates from the premise that core SACL is already in place, providing RFC-1918, RFC-2267 and bogon (IANA unassigned IPv4 address space) filtering. The purpose of the local additions is to provide a way to add additional filter rules after the anti-smurf rules are applied. For example, if you wanted to add ICMP filtering rules, you would add the following SACL lines to the local-additions file:

```
! Local-Additions  
deny icmp any any fragments log-input ! Attack Exploit  
deny icmp any any echo log-input ! Disables remote ping  
deny icmp any any echo-reply log-input ! Disables local ping  
deny icmp any any net-redirect log-input ! Attack Exploit  
deny icmp any any host-redirect log-input ! Attack Exploit  
permit icmp any any source-quench  
permit icmp any any ttl-exceeded  
permit icmp any any unreachable  
deny icmp any any log-input ! drops all other ICMP messages
```

The rules are then processed by the script adding and subtracting rules as needed.

```
# Define the location of the local additions file.  
LOCADD="local-additions"  
#  
# These variables do not need to be edited, but you can if you want
```

```
IGWA="$1"  
DESTNET="$2"  
DESTMSK="$3"  
ACL=$TFTP/namedacl-update  
DELACL=$TFTP/namedacl-del  
RUNDATE=`cat /var/tmp/rundate`  
RMSmurf="./acl-edit-rm.exp $RTR"  
INSsmurf="./acl-edit.exp $RTR"  
#  
# Make sure you have the placeholder files  
/usr/bin/touch $ACL  
/usr/bin/touch $DELACL  
#/usr/bin/touch /var/tmp/rundate
```

```
# Command Line Syntax Checking
```

```
if [ "$1" = "" ]  
then  
echo "Router External Interface Address Missing";exit  
fi
```

```
if [ "$1" = "-h" ]  
then
```

Revised October 17, 2002

Page 7 of 12

# Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

```
echo "Command Syntax:
First Field <Router Ext Addr> (If Addr is assigned via DHCP use 0.0.0.0)
Second Field <Local Network Prefix> (Network Prefix in dotted Quad)
Thrid Field <Local Prefix Mask> (Enter mask using Cisco's wildcard
format)";exit
fi

if [ "$2" = "" ]
then
echo "Local Network Prefix Missing";exit
fi

if [ "$3" = "" ]
then
echo "Wildcard Mask Missing (Cisco Wildcard Format)";exit
fi

/usr/bin/clear

# Here we archive the last runs delete file
touch $TFTP/namedacl-update
touch $TFTP/namedacl-del
cp $TFTP/namedacl-del $ARCHDIR/namedacl-del-previous
#
# Here we create the archive of the last runs add file (needed for diff
report)
cp $TFTP/namedacl-update $ARCHDIR/namedacl-previous
```

You may recall that while named access lists permit editing to the extent of rule removal, rule reordering is not permitted. When a rule is deleted, the rules below shift up. If new rules are added, they are appended to the end of the list. For the list to be edited properly, all of the SACL rules from the first anti-smurf rules must be deleted. The config-eng Expect script called `acl-edit-rm.exp` performs this function.

```
# Call Expect Delete Loader
echo
echo Removing Previous smurf filter from $RUNDATE
echo
$RMsmurf
#
# Now we delete the previous runs retraction and addition files
rm $DELACL
rm $ACL

#
# The basis of our smurf defense filter is provide by Oystein Homelien and
# PowerTech Information Systems (AKA the Smurf Amplifier Registry)
# in Oslo, Norway.
#
echo "Getting File smurf Amplifire List....."
wget -O /var/tmp/acl-raw-smurf
http://www.powertech.no/smurf/list.cgi?format=cisco-acl
```

# Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

```
#  
/usr/bin/clear
```

This section handles the SED (streaming editor) processing of the PowerTech ACL. It converts it from the standard Cisco access list format to named access list format. Two lists are actually created: the new SACL additions called \$TFTP/namedacl-update and a retraction list called \$TFTP/namedacl-del for the newly created additions list. The retractions list is called by the script the next time the script is run to delete the previous runs additions.

```
#  
echo "Processing File..."  
grep "access-list" /var/tmp/acl-raw-smurf > /var/tmp/acl-clean-p1  
sed -n 's/<p><pre>//p' /var/tmp/acl-clean-p1 > /var/tmp/acl-clean-p2  
sed 'ld' /var/tmp/acl-clean-p1 > /var/tmp/acl-clean-p3  
cat /var/tmp/acl-clean-p2 /var/tmp/acl-clean-p3 > /var/tmp/acl-clean-p4  
#  
  
echo "Formating File's..."  
  
# Formatting the raw ACL from Powertech  
awk '{print $4,$5}' /var/tmp/acl-clean-p4 > /var/tmp/acl-format-p1  
sed 's/.*\/deny ip & any/g' /var/tmp/acl-format-p1 > /var/tmp/acl-format-p2  
  
# Formating the new addition list  
echo $aclname > /var/tmp/acl-name  
echo "no permit ip any $DESTNET $DESTMSK" >> /var/tmp/acl-name  
echo "no permit ip any host $IGWA" >> /var/tmp/acl-name  
cat /var/tmp/acl-name /var/tmp/acl-format-p2 > /var/tmp/acl-aggregate  
  
echo "Checking For Local Additions File"  
# Local Additions section.  
if [ `ls $APD | grep -c $LOCADD` = "1" ]; then  
cat $LOCADD >> /var/tmp/acl-aggregate ;  
else  
echo "No local additions file"  
fi  
  
echo "permit ip any $DESTNET $DESTMSK" >> /var/tmp/acl-add  
echo "permit ip any host $IGWA" >> /var/tmp/acl-add  
cat /var/tmp/acl-aggregate /var/tmp/acl-add > $ACL  
  
# Formatting the new retraction file  
if [ `ls $APD | grep -c $LOCADD` = "1" ]; then  
cat $LOCADD >> /var/tmp/acl-format-p2  
fi  
sed 's/.*\/no & /g' /var/tmp/acl-format-p2 >> /var/tmp/acl-rmlist  
echo $aclname > $DELACL  
cat /var/tmp/acl-rmlist >> $DELACL  
Once both lists have been created, the new SACL additions are copied to the  
router using the config-eng script called acl-edit.exp.
```

# Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

```
echo
echo Installing Updated `date +%b-%d` smurf filter
echo
$INSSmurf
# Removing ACL temp files
rm -rf /var/tmp/acl*
#
echo Archiving The Addition and Retraction List
mkdir $ARCHDIR > /dev/null 2>&1
cp $ACL $ARCHDIR/namedacl-update-`date +%b-%d`
cp $DELACL $ARCHDIR/namedacl-del-`date +%b-%d`
date +%b-%d > /var/tmp/rundate
/bin/chmod ug-rw,o+rx $ACL
/bin/chmod ug-rw,o+rx $DELACL
echo
echo
echo The Router Public Interface is $IGWA
echo Inbound traffic is permitted for hosts in prefix $DESTNET
echo The Update File $ACL has been created.
echo The retraction file $DELACL has been created.
echo
echo
#
```

When the script is complete, it provides a simple report on what was added from local additions and reports the location of the addition and deletion files. Since the function is best run as a cron job, it is best to create a script wrapper that can be called from cron. You may have noticed in the sample directory listing above a file called igw-wrapper.cron. This file is the wrapper for updating the router called igw, here is the wrapper:

```
#!/bin/sh
# This is a simple script wrapper all output is reported to mail.
# Enter the script location here:
SCR="/usr/local/smurf-acl/smurf-filter.sh"
RGW="172.30.48.2"
LNET="168.74.20.0"
LMSK="0.0.0.255"
MRCPT=admins@any.net.com

$SCR $RGW $LNET $LMSK | mail $MRCPT
#
#
```

Below is an edited version of the config-eng Expect script to upload the SACL addition and removal files. To run the script in its default form, a script file called acl-edit.exp and acl-edit-rm.exp need to be in the local directory with the smurf-filter.sh script. Uncomment the appropriate copy command and save the file with the correct name. You will also need to enter the target router address, username/password and enable/password variables for the script to run properly. The scripts are called by the smurf-filter.sh, so no cron wrapper is needed.

# Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

```
#!/usr/local/bin/expect
# Copyright (C) 2002 Michael J Martin.
#
# command syntax <host ip addr> <tftp server addr> (if CLI options enabled)
#
# Administrator Defined Variables
#
# If you want to define the target router using the CLI uncomment (good
option
# if you use the tool as a CLI tool
#
# set host [lindex $argv 0]
# If you want to define the target host within the script uncomment (good
option
# if you are calling this script from CRON)
#
# set host x.x.x.x.
#
# If you want to define the TFTP server as a CLI option comment
#set TFTPS [lindex $argv 1]
#
# If you want to define a static TFTP server uncomment
# set TFTPS 172.30.71.7
#
# Username Properties
#
# Set the Enable password here
# set ENPASS enable
# Set the username here, needed if TACACS/RADIUS authentication is used on
# the routers
# set USER config
# Set the username password or VTY password here
# set PASS config
#
# Set how you want to connect to the router Telnet or SSH
spawn telnet $host
#spawn ssh -l $USER $host
#
# Login into the router
#
expect "Username:" {send "$USERr"}
"Password" {send "$PASSr"}
"refused" exit
#
expect "Password" {send "$PASSr"}
">" {send "enabler"}
#
# Enter into Enable Mode
#
expect ">" {send "enabler"}
"Password:" {send "$ENPASSr"}
Revised October 17, 2002
```

# Smurf Fraggle Attack Defense Using SACLs

Michael J. Martin

```
#
expect "Password:" {send "$ENPASSr"}
"#" {send "r"}
#
#
# Upload the configuration
#
expect "#
#
# To run as acl-edit.exp uncomment below:
send "copy tftp://$TFTPS/namedacl-update runr"
#
# to run as acl-edit-rm.exp uncomment below
#send "copy tftp://$TFTPS/namedacl-del runr"

expect "g]?"

send "r"

expect "#

send "copy run startr"

expect "ig]?"

send "r"

expect "#

send "logoutr"
```

## Conclusion

This closes the Defending the Homeland series. I hope that you found these articles helpful and relevant to some of your intrusion detection challenges. If there is a topic or solution that you would like to hear about, please let the site editor know. Next month, we'll cover BGP basics or IOS VLAN solutions for using WLAN securely, so stay tuned. On a personal note, I would like to thank my wife, Sarah, for enduring one of the worst pregnancies in history with strength and grace and bringing my new son, Samuel Philip Martin, into the world.