Practice Builders - Set Up an Office Proxy Server to Limit Staff Web Surfing

## James M. Nachbar, MD, FACS, April 28, 2014

# Two main sources of viruses and malware in offices:

1) Email attachments

2) Surfing to websites

Also, surfing is a huge time-waster. But, you often can't completely cut off access to all web servers, but typically there are a few you need them to be able to use (e.g. Allergan's Brilliant Distinctions)

The solution is a Proxy Server. The browsers on your staff's computers point to the proxy server, so all requests route through the proxy server. The proxy server has rules you have defined which determine which websites are allowed. Can be different for different computers.

Most effective, and recommended, is a network configuration where direct access to websites is blocked by the firewall, so if the browser configuration is changed, or another computer connected, they can't bypass the Proxy Server. However, you can set up a Proxy Server, even outside the office, to try it out, even if you don't have a firewall that blocks outgoing traffic. It is also possible to use a Proxy Server outside the office for your regular operations, and if you get a service, you won't have to learn Linux. However, if you are shopping for a service, be aware that most "Proxy Server Services" are focused on keeping your identity private (since requests don't come from your own server), and allow access to any website. Our focus is going to be on limiting the domains that can be browsed.

**EVERYTHING I AM GOING TO RECOMMEND IS FREE SOFTWARE. HOWEVER, IF YOU USE AMAZON WEB SERVICES (AWS) TO SET UP A TRIAL SERVER, THERE MAY BE CHARGES FROM AWS.**

My configuration uses an inexpensive computer running Ubuntu Server LTS (version 14.04 was released this month, and will have security updates for five years). This is a version of Linux, so you will need to know a little about Linux to do this, but even if you don't run your own servers, it helps to understand what to ask your IT person to set up.

The server acts as firewall, as well as proxy server, and also restricted webmail server (running a modified SquirrelMail server which lets staff read their emails, but is modified NOT to pass them the attachments). (Also acts as DNS server and DHCP server)

The server has TWO ethernet connections. One goes to the Cable Modem, and the other goes to the switch that the office computers are on.

The firewall is configured using iptables, which allows me to control which services the computers can access. There are lots of sources for iptables, and other ways to configure the firewall, but this is what I use.

# My iptables configuration script looks something like this.

The iptables commands add rules to specific named "chains". The rules are followed in order until a packet matches. When it does, the instruction for that rule is followed,

for example going to a different chain (-j), including the named chains of ACCEPT and DROP which tell the server what to do with that packet

I run this script with sudo, like:

sudo ./firescript

# JMN - run this script as root (/sbin on PATH), then run iptables-save to save result

# to change PATH: # PATH=/sbin:$PATH then export PATH

# in /etc/sysconfig: /sbin/iptables-save > iptables

#

# JMN - commented out logging, since I don't look at logs, and

# perhaps the logging is causing computer to occasionally crash

# JMN - The command to track ongoing connections from the local net

# was also jumping them directly to ACCEPT, without regard for the RULES.

# Thus, so that I can prevent some, but not all, local traffic from reaching

# the net, I create a chain, ALLOWOUT, that jumps to ACCEPT as it tracks

# the ongoing connections.

# In order for forwarding to work, the kernel needs to be told to allow forwarding among interfaces:

sysctl -w net.ipv4.conf.all.forwarding=1

# Here, tell iptables which interface is EXTernal (to the Internet), and which is LOCAL

EXT=eth0

LOCAL=eth2

# Define the local, private network. the /24 indicates that the fourth octet (0) can be anything, so this will match 192.168.1.12 and 192.168.1.213

PRIVATE=192.168.1.0/24

# The credit card machine -- it needs to be able to connect to its servers

CREDITCARD=192.168.1.118

# another server on my network, which I want to be able to access from the outside

PRINGLE=192.168.1.92

# The portion of the subnet that I want to be able to access the Internet directly. with the /28, this will match 192.168.1.208 - 192.168.1.254

PRIVEXTACCESS=192.168.1.208/28

SPECIALCOMPUTERALLOWED=192.168.1.135 # a special computer that might need temporary access

#default policies. We allow outgoing traffic from processes on THIS server (OUTPUT), but deny all traffic from other servers (unless otherwise allowed) and block all incoming traffice (also unless explicitly allowed)

iptables -P INPUT DROP

iptables -P OUTPUT ACCEPT

iptables -P FORWARD DROP

#flush the chains, including any references to non-builtin chains

iptables -F

#flush the nat table, too, since testing may have added entries we don't want anymore

iptables -t nat -F

#delete any non-builtin chains

iptables -X

# log all traffic from PRINGLE (commented out). Unlike the other chains, LOG just logs the packet but does not move to a new chain

#iptables -A INPUT -s $PRINGLE -j LOG

#make a new chain for all the input and forwarding rules

iptables -N RULES

#make a new chain on the way out so ongoing connections can be tracked

iptables -N ALLOWOUT

#keep state of locally and subnet-generated connections These are added to chains for which traffic is allowed out

iptables -A OUTPUT -m state --state NEW -o $EXT -j ACCEPT

iptables -A ALLOWOUT -m state --state NEW -s $PRIVATE -o $EXT -j ACCEPT

# We are restricting outbound traffic from local subnet,

# but allow subnet traffic connecting to this server (e.g., for mail relay

# for scanner)

iptables -A INPUT -i $LOCAL -j ACCEPT # ONLY GOES TO LOCAL MACHINE, NOT FORWARDING

# Forward all remaining traffic to single RULES table, for uniform management.

# Forward each interface separately, so we can compare traffic used by each interface, since each packet routed by each line in this file is counted

iptables -A INPUT -i $EXT -j RULES

iptables -A INPUT -i $LOCAL -j RULES

# Now, catch-all for any missed

iptables -A INPUT -j RULES

iptables -A FORWARD -i $LOCAL -j RULES

# now, incoming traffic - segregate by where it is going

iptables -A FORWARD -o $LOCAL -j RULES

# any missed coming in

iptables -A FORWARD -i $EXT -j RULES

# catch-all, so everything winds up going to RULES

iptables -A FORWARD -j RULES

#for testing

#iptables -A RULES -j LOG

#iptables -A RULES -j ACCEPT

#iptables -A RULES -i $LOCAL -j ACCEPT

# (uncomment the following to allow local net out)

#iptables -A RULES -i $LOCAL -j ALLOWOUT

# This is how to allow access from any of your local computers to a specific server. Note that you need IP addresses rather than DNS names, for performance reasons.

# You can use DNS names in your Proxy Server, if needed

# For stamps.com

#iptables -A RULES -i $LOCAL -d 216.52.211.73 -j ALLOWOUT # auto.internetpostage.com

#iptables -A RULES -i $LOCAL -d 216.52.211.57 -j ALLOWOUT # ams.internetpostage.com

#iptables -A RULES -i $LOCAL -d 216.52.211.52 -j ALLOWOUT # postage.stamps.com

#iptables -A RULES -i $LOCAL -d 216.52.211.58 -j ALLOWOUT # postage.internetpostage.com

#prevent external packets from using loopback

iptables -A RULES -i $LOCAL -s 127.0.0.1 -j DROP

iptables -A RULES -i $EXT -s 127.0.0.1 -j DROP

iptables -A RULES -i $LOCAL -d 127.0.0.1 -j DROP

iptables -A RULES -i $EXT -d 127.0.0.1 -j DROP

#anything coming from internet should have a real IP address

iptables -A RULES -i $EXT -s 192.168.0.0/16 -j DROP

iptables -A RULES -i $EXT -s 172.16.0.0/12 -j DROP

iptables -A RULES -i $EXT -s 10.0.0.0/8 -j DROP

#block outgoing NETBIOS

iptables -A RULES -o $EXT -p tcp --sport 137:139 -j DROP

iptables -A RULES -o $EXT -p udp --sport 137:139 -j DROP

iptables -A RULES -o $EXT -p tcp --sport 445 -j DROP

# allow outgoing traffic on a specific TCP port to a specific server

iptables -A RULES -p tcp --dport 61614 -d 50.16.31.249 -j ACCEPT

#allow NTP packets from private subnet

iptables -A RULES -i $LOCAL -p udp --dport 123 -j ACCEPT

#allow DNS packets from private subnet

iptables -A RULES -i $LOCAL -p udp --dport 53 -j ACCEPT

iptables -A RULES -i $LOCAL -p tcp --dport 53 -j ACCEPT

# For GoToMeeting and GoToAssist:

iptables -A RULES -i $LOCAL -p tcp --dport 8200 -j ACCEPT

#allow outgoing from credit card terminal

iptables -A RULES -i $LOCAL -s $CREDITCARD -j ALLOWOUT

#allow outgoing from PRINGLE

iptables -A RULES -i $LOCAL -s $PRINGLE -j ALLOWOUT

#allow outgoing from priviledged sites

iptables -A RULES -i $LOCAL -s $PRIVEXTACCESS -j ALLOWOUT

# uncomment this line to allow SPECIALCOMPUTERALLOWED to access the Internet temporarily

# iptables -A RULES -i $LOCAL -s $SPECIALCOMPUTERALLOWED -j ALLOWOUT

# allow loopback

iptables -A RULES -i lo -j ACCEPT

#allow icmp. This will allow outside computers to "ping" your server, which you may or may not want to allow

iptables -A RULES -p icmp --icmp-type any -j ACCEPT

# uncomment to allow incoming ftp

#iptables -A RULES -i $EXT -p tcp --dport 21 -j ACCEPT

#rules for incoming packets from LAN

#allow continuation of existing connections

iptables -A RULES -m state --state ESTABLISHED,RELATED -j ACCEPT

#allow SSH to be set up

# consider using a non-standard port for ssh, to reduce attacks by scripts

iptables -A RULES -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT

# This rule will allow access only from the inside, if you want to differentiate

iptables -A RULES -i $LOCAL -m state --state NEW -p tcp --dport 22 -j ACCEPT

# allow inbound traffic on 2080 for http to PRINGLE

iptables -A RULES -m state --state NEW -m tcp -p tcp --dport 2080 -j ACCEPT

#allow ftp to be set up

#(don't forget to modprobe ip\_conntrack\_ftp)

iptables -A RULES -m state --state NEW -m tcp -p tcp -i $LOCAL --dport 20 --modprobe=ip\_conntrack\_ftp -j ALLOWOUT

iptables -A RULES -m state --state NEW -m tcp -p tcp -i $LOCAL --dport 21 -j ALLOWOUT

# allow packets to be forwarded to PRINGLE:80 and :22

iptables -A RULES -d $PRINGLE -p tcp --dport 80 -j ACCEPT

iptables -A RULES -d $PRINGLE -p tcp --dport 22 -j ACCEPT

#default for RULES:

iptables -A RULES -j DROP

# uncomment for testing:

#iptables -A RULES -j LOG

#iptables -A RULES -j ACCEPT

#masquerade connections - this will set up NAT, so appropriate connections will be forwarded through this server,

# replacing the source IP address from the internal network with the external IP address of the server,

# so that return traffic will come back to this server, for routing back to the internal source computer, if the firewall allows

iptables -t nat -A POSTROUTING -s $PRIVATE -o $EXT -j SNAT --to-source $EXTIPJMN

# Forward port 2080 to pringle:80 (port 80 on PRINGLE)

iptables -t nat -A PREROUTING -d $EXTIPJMN -p tcp -i $EXT --dport 2080 -j DNAT --to-destination $PRINGLE:80

# Proxy Server Program – Open Source “Squid” Proxy Server

Then, the Proxy Server program, running on the firewall server, can be accessed by the computers on the local network, and the Proxy Server can access the Wild Internet. So, sites allowed by the Proxy Server will be available, and other sites will not.

**The Proxy Server I recommend is Squid, version 3.X. This is a free, open source proxy server, that is very full featured (we will only be using a tiny portion of those features)**

**To try out the Proxy Server**, I recommend using Amazon Web Services. This will allow you to run a computer in the cloud for 2 cents per hour, with a minimum of one hour (no kidding!). And, for the first year, you can run a "micro" server for free. Google Amazon Web Services, and sign up for a free account. Go to the "EC2" dashboard to control your cloud servers.

You will need to set up ssh access from your computer to control the Amazon Web Services computer. For Macs, this is very easy -- run the Terminal program, and ssh is already installed! For Windows computers, you have to install an ssh client. I recommend Putty, a free ssh client that will work on Windows computers.

You will also need to create a "key pair" to be able to access your AWS instances. A "key pair" is a combination of public and private keys that work together. As the name states, you can distribute the public key anywhere you want, but you MUST keep the private key private, because anyone who gets the private key can access any of your computers. On the Mac, you can generate a key pair by typing:

openssl genrsa -des3 -out private.pem 2048

openssl rsa -in private.pem -outform PEM -pubout -out public.pem

If you remove the -des3, you will not have to add a passphrase, which will be required every time you use your private key. However, not using a passphrase allows anyone who finds your private key file to use it.

This generates the private key first. The public key can be derived from the private key, which is what the second command does.

Use any text viewer program to check the contents of the files. Make sure you know which one contains only the public key. You can Import the public key from the "Key Pairs" link in the EC2 dashboard.

If you don't want to keep your private key from ever leaving your local computer, you can use the "Create Key Pair" on the Key Pairs page of the EC2 dashboard. Make sure you copy the private key to your local computer, since it won't be available after you close the window from generating the key pair.

Put the private key where your ssh client can find it, and it will allow you to access your AWS server. On the Mac, that is a file called id\_rsa within the .ssh folder in your home folder. The permissions of that file must prohibit access from group and other, and must be owned by you, or ssh will refuse to use it (chmod 600 id\_rsa)

Once you have created your key pair, you can start your Squid instance by following these steps:

1) In the EC2 dashboard, Instances page, click "Launch Instance"

2) Find Ubuntu Server 14.04 LTS image. This is the starting configuration of your server.

3) Make sure "64 bit" is selected, and click on "Select" next to your server AMI

4) Pick the instance size. Choose "Micro", which is 2 cents per hour, and is more than capable enough to run the Squid Server, especially for testing

5) Click "Next: Configure Instance Details"

6) Review the settings on the Configure Instance Details page, but the defaults should be acceptable. Click "Next: Add Storage"

7) Review the storage options. The defaults should work, but note that this includes "Delete on Termination", which means that your virtual disk drive will be deleted when you terminate your instance. That is probably better than saving it unless you want to re-use it or do something with it.

8) Click "Next: Tag Instance"

9) Click "Next: Configure Security Group"

The Security Group is like an external firewall that AWS sets up for you. You want the least amount of access, to reduce the risk that your server will be attacked.

10) select "Create a new security group", give it a name and description if you want, and add two rules. The first is Type "SSH", which will select Protocol TCP, and Port range 22. For "Source", select "My IP". That will prevent access from any computer not on the same router that you are on.

11) Add a second rule (click "Add Rule"), type Custom TCP Rule, Protocol TCP, Port Range 3128, Source My IP. This is the port that the Proxy Server will be listening on for connections from your browser.

12) Click "Review and Launch". You can review the selected options, and then click "Launch".

13) You will then be presented with a dialog to select or create a key pair. Select the key pair you imported previously. This is the key pair that will work with your new server.

14) Check the acknowledgement box, and click "Launch Instances"

15) You can review the information on that page, but then click "View Instances"

16) It will take about five minutes for your Instance to spin up and become live. If you click on your new instance in the list at the top of the "Instances" page, you will see the details for that instance at the bottom of the page. You can look at Status Checks, or just wait a couple of minutes. Note that you will need the "Public IP" to connect to that server. Make a note of that Public IP address.

17) Go to your ssh client, and try to access your new server, using the "Public IP" as the address, user name "ubuntu", and the key pair you set previously. For security reasons, it is not possible to log into the server with a username and password. Only the key pair will work. From the Mac's Terminal window, if the Public IP is 54.82.54.87, type ssh ubuntu@54.82.54.87

18) You may get a message that connection times out or is refused if the server is still booting. Once it succeeds, you will get a prompt asking if you want to "continue connecting", saving the new server information to your computer. On subsequent connections to the same computer, you will not get that prompt. If you happen to get the same IP address as you had used previously for a different server, the fingerprint will be different, and you will not be able to connect. You will be told which line in the .ssh/known\_hosts line is the problem, so delete that line.

19) Once you agree to continue connecting by typing "yes", if your keys are set up properly, you will be connected.

20) Next, you must update your server to the most current version of its packages, for security reasons. Type: sudo apt-get update to get info about the new packages, and then sudo apt-get upgrade to actually do the upgrades. Type "y" to confirm. The whole upgrade will only take a couple of minutes, since there is not much software in your new server.

21) Install Squid by typing "sudo apt-get install squid3". This will download, install, and start Squid.

22) Now, check that your new Squid is working by setting up your Internet Explorer or other browser to use it to connect to the Internet. For Internet Explorer, select "Tools/Internet Options", go to the Connections tab, click on LAN Settings, and in the bottom of the page, check the box that says "Use a Proxy Server", enter the IP address of your new server (like 54.82.54.87), and port 3128, and click OK twice to save your new settings.

23) Now, use your browser to go to any http site, like http://www.google.com . If everything is working correctly, you will get a big "ERROR" page, saying that Access Denied. This means that your Proxy Server is working properly, and rejecting your attempt to connect to www.google.com (which actually is a dangerous website, since ANYTHING can be found on Google). If you don't get Access Denied, recheck your Security Group rules that port 3128 is allowed in the Inbound Rules

24) Now, you need to configure your Squid Server to allow access to the sites you want to allow. You need to set BOTH the "incoming" and "outgoing" sites that are allowed, so that only your own browser can access the Squid server, not any browser anywhere in the world (although your Security Group would prevent that, anyway). This will require editing the Squid Configuration file. My favorite Linux editor is "vi", but you may find "nano" easier, since it has prompts at the bottom of the page. You will need to run nano with sudo, so that you will have permission to edit the configuration file.

25) Type "sudo nano /etc/squid3/squid.conf" to start nano. (You might need to use "sudo nano /etc/squid3/squid3.conf", depending on your version of Squid)

26) Lines beginning with "#" are comments, and have no effect. Find the line that says http\_access allow localhost (not commented out). After that line, type "acl my\_ip src 12.23.34.45", where you replace 12.23.34.45 with the IP address of your router (it will be shown in the lines for the Security Group). Then add a line "acl my\_dest dstdomain .google.com" . The .google.com will allow any site ending in .google.com to be accessed. You can add additional allowed sites and domains after .google.com, each separated by a space. As noted above, you do NOT want .google.com in your production proxy server, but it is helpful for testing

27) Immediately after that add a line "http\_access allow my\_ip my\_dest". That MUST be before the http\_access deny all, which prevents any requests not explicitly allowed from completing.

28) If you want to allow regular (http://) and secure (https://) access to the sites you select, find the line that says "http\_access deny CONNECT !SSL\_ports", and comment out (add a '#' sign before the first character)

29) Reload the Squid configuration by typing "sudo squid3 -k reconfigure". If everything is OK, it will execute silently. If you made a mistake, you will get a helpful message about what to do.

30) Now, try accessing http://www.google.com from your browser. It should succeed. Try browsing anywhere else, and you will get the ERROR Access Denied page.

31) https will work, too. Browse to https://www.google.com to see. However, if the page is not allowed, you may get a less helpful error message from Internet Explorer. However, if you click "Fix Connection Problems", you will see a message that you got error message 403 (Forbidden) to let you know what the problem is.

32) Type "sudo nano /var/log/squid3/access.log" to view the access log. This will show TCP\_MISS for ALLOWED sites, and TCP\_DENIED for denied sites. If you are trying to connect to a new server and entering its domain into dstdomain doesn't work (don't forget sudo squid3 -k reconfigure), look in the access.log to see if the site is trying to get you to go to a different site, and make a decision about whether to allow that site, as well. The same applies if the website looks awful - some of the files needed to show the website properly may be hosted in a different domain. Checking the log will tell you what other sites you need to open to show the site correctly.

33) When you are done testing, don't forget to terminate your instances, since you will be charged by the hour until they are terminated.