Security and network design

Remco Hobo

January 18, 2005

Nessus scan of own system

Nessus is a program which can scan a computer for vunerabilities. It uses a unix server to scan from. The client, which can be used to view results and to configure the server can be a windows platform. After the scan of my computer was completed, the following weaknesses where found:

- The remote host is using a version of mod_ssl which is older than 2.8.18.

  This version is vulnerable to a flaw which may allow an attacker to disable the remote web site remotely, or to execute arbitrary code on the remote host.

  *** Note that several Linux distributions patched the old version of *** this module. Therefore, this alert might be a false positive. Please *** check with your vendor to determine if you really are vulnerable to *** this flaw

  Solution : Upgrade to version 2.8.18 or newer Risk factor : Low

- The remote host is vulnerable to an 'Etherleak' - the remote ethernet driver seems to leak bits of the content of the memory of the remote operating system.

  Note that an attacker may take advantage of this flaw only when its target is on the same physical subnet.

  See also : http://www.atstake.com/research/advisories/2003/a010603-1.txt Solution : Contact your vendor for a fix Risk factor : Serious

After browsing the internet, the first flaw brings up the following information:

```
SYNOPSIS

mod\_ssl (www.modssl.org) is a commonly used Apache module that
 provides strong cryptography for the Apache web server. The
 module utilizes OpenSSL (formerly SSLeay) for the SSL implementation.
 modssl versions prior to 2.8.7-1.3.23 (Feb 23, 2002) make use of the
 underlying OpenSSL routines in a manner which could overflow a buffer
 within the implementation. This situation appears difficult to
```

```
 exploit in a production environment, however, for reasons detailed
 below.

CAUSE

The session caching mechanisms utilizing dbm and shared memory
 utilize the OpenSSL routine i2d\_SSL\_SESSION, which "serializes"
 an SSL session into a format that can be stored in the session cache.
 The OpenSSL docs inform us:

    When using i2d\_SSL\_SESSION(), the memory location pointed to by pp
     must be large enough to hold the binary representation of the session.
     There is no known limit on the size of the created ASN1 representation,
     so the necessary amount of space should be obtained by first calling
     i2d\_SSL\_SESSION() with pp=NULL, and obtain the size needed, then
     allocate the memory and call i2d\_SSL\_SESSION() again.

mod\_ssl < the version listed above do not do this, however, and could
 potentially lead to an overflow of the static buffer used by mod\_ssl
 for holding the contents of the serialized session.



DETAILS


An example of the relevant mod\_ssl source is listed below:


(mod\_ssl < 2.8.7) (www.modssl.org)
 ssl\_util\_ssl.h:
 \#define SSL\_SESSION\_MAX\_DER 1024*10
  ssl\_scache\_dbm.c:
  BOOL ssl\_scache\_dbm\_store(server\_rec *s, UCHAR *id, int
               idlen, time\_t expiry,SSL\_SESSION \*sess) \{

 <snip>
  UCHAR ucaData[SSL\_SESSION\_MAX\_DER];
 <snip>
  ucp = ucaData;
  nData = i2d\_SSL\_SESSION(sess, \&ucp);
```

This weakness, the mod_ssl was easily fixed by executing ”emerge mod_ssl”.

The second weakness, the ’Etherleak’, means that the card seems to leak bits of system memory when doing certain operations. The article attached to the Nessus report, advices to contact the vendor of the NIC. Unfortunately there are no updated drivers available for this network card. This vunerability is not curable at this time. One work-around is to deny all ICMP traffic from and to the server, since the vunerability is most effective on the ICMP traffic.

Nessus scan of colleague's system

After checking my own computer, I checked Wouter Borremans's computer for flaws.

He also has the NIC flaw, but also has the following weakness:

- You are running a version of OpenSSH which is older than 3.7.1

  Versions older than 3.7.1 are vulnerable to a flaw in the buffer management functions which might allow an attacker to execute arbitrary commands on this host.

  An exploit for this issue is rumored to exist.

  Note that several distribution patched this hole without changing the version number of OpenSSH. Since Nessus solely relied on the banner of the remote SSH server to perform this check, this might be a false positive.

  If you are running a RedHat host, make sure that the command : rpm -q openssh-server

  Returns : openssh-server-3.1p1-13 (RedHat 7.x) openssh-server-3.4p1-7 (RedHat 8.0) openssh-server-3.5p1-11 (RedHat 9)

With this vunerability, it is in theory possible to produce a buffer overflow afterwards arbitrary code can be executed. There is no proof-of-concept available so the risk factor is relatively low.

This flaw was easily fixed executing "emerge openssh".

Nessus scan of own system by colleague

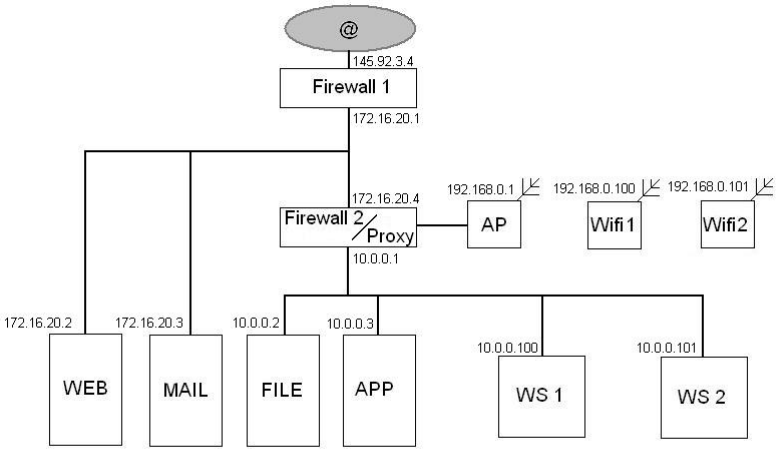
My colleague used the same scanning methods as I have used myself and came up with the same results.

Design a network for a virtual company

The last assignment was to design a network layout for a small company/institute. It should contain

- a web server (accessible from the internet)
- a mail server (accessible from the internet)
- a file server
- an application server
- some client
- some wireless client

We had to make a network diagram and the network policies. The final network layout will look like:



The policies we have set up specifically for the network are:

- Firewall 1 will filter all traffic from the outside and will only allow
  - HTTP
  - HTTPS
  - IMAPS
- Firewall 2 will block all outside traffic
- Normal users will only have browsing and mail internet functionality, everything else has to be cleared by the IT department. Users will be authorised on the proxy server.
- Wifi user will have to use the 802.x secure protocoll and can not connect to any internal server.

Beneath is an example firewall script for firewall 1

```
#!/bin/bash -x

IT=/sbin/iptables

WEBSERVER_IP=172.16.20.2
MAILSERVER_IP=172.16.20.3
INTERNET_IP=145.92.3.4
LOCAL_IF=eth1
INET_IF=eth0

$IT --flush
$IT -t nat --flush
```

```
$IT -t mangle --flush


echo 0 > /proc/sys/net/ipv4/ip_forward


# BLOCK everything so we can peacefully fill the chains
$IT --append INPUT --jump DROP
$IT --append FORWARD --jump DROP
$IT --append OUTPUT --jump DROP

# Local traffic is always ok
$IT --append INPUT --in-interface lo --jump ACCEPT

# We can poke around and get response
$IT --append INPUT --match state --state ESTABLISHED,RELATED --jump ACCEPT

# General rule: allow what has already been approved
$IT --append FORWARD --match state --state ESTABLISHED --jump ACCEPT

$IT --append FORWARD --match state --state RELATED --jump ACCEPT


#Allow traffic from local interfaces
$IT --append INPUT --in-interface $LOCAL_IF --jump ACCEPT

#Outgoing internet chain
$IT --new-chain ToInternet
$IT --append FORWARD --out-interface $INET_IF --jump ToInternet


# Masquerade out to the internet.
$IT --append POSTROUTING -t nat --out-interface $INET_IF --jump MASQUERADE


#incomming internet chain
$IT --new-chain FromInternet
$IT --append FORWARD --in-interface $INET_IF --jump FromInternet
$IT --append INPUT --in-interface $INET_IF --jump FromInternet


#IMAPS
$IT --append FromInternet --protocol tcp --destination-port imaps --match ->
state --state \newline NEW --jump ACCEPT
$IT -A PREROUTING -t nat -p tcp -d $INTERNET_IP --dport imap -j DNAT --to ->
$MAILSERVER


#http
$IT --append FromInternet --protocol tcp --destination-port 80 --match ->
```

```
state --state NEW --jump ACCEPT
$IT -A PREROUTING -t nat -p tcp -d $INTERNET_IP --dport 80 -j DNAT ->
--to $WEBSERVER

#https
$IT --append FromInternet --protocol tcp --destination-port 443 --match ->
state --state NEW --jump ACCEPT
$IT -A PREROUTING -t nat -p tcp -d $INTERNET_IP --dport 443 -j DNAT ->
 --to $WEBSERVER


#drop non matching trafic
$IT --append FromInternet --jump DROP

# Allright, now start it up
echo 1 > /proc/sys/net/ipv4/ip_forward
$IT --delete OUTPUT 1
$IT --delete INPUT 1
$IT --delete FORWARD 1
```

References

[1] http://www.nessus.org, The Nessus website