REMOTE TIMING TECHNIQUES

Introduction

This paper describes remote timing techniques based on TCP/IP intrinsic operation and options. The techniques are used for careful observation of the TCP/IP data stream to detect timing differences in the operation of the remote application and relate them to selected data and/or phenomena.

Basics

The methods will be made clear with a practical example, developed in this paper.

Suppose that we want to know if a given username exists or not in a remote system. What we can attempt to do is to carefully look for processing timing differences in the remote logon process, between the path taken by the remote application when a given username exists, and when it doesn't. If we find a statistical difference (no matter how small) between the two instances, we can determine if the username exists or not.

For this method to work, some conditions must be met. First of all, a relatively appreciable difference in the processing times of the two different processing paths must exist.

If this difference exists, the problem now is, how can we get a detailed estimation of the remote processing times of each processing path, given such facts as variable network latency and packet loss, variable loads on the remote system, local factors, and such.

Causes of Appreciable Processing Time Differences

The initial cause of timing differences must be, of course, the application's different processing paths itself. Now, for this difference to become appreciable, some slow event must occur in one of the paths and not in the other. A typical "slow event" in a computer, is disk access. Other main cause of "slowness" is the computer itself. The examples in this paper were tested on relatively "slow" (old) computers, and could not be reproducible in all environments.

Remote Timing Techniques (RTT)

Timestamps

Timestamps were added to TCP to allow precise estimation of the Round Trip Time (RTT), which is necessary on high bandwidth networks to avoid data loss and/or congestion. As a side effect, they also allow (at least in some implementations), "precise" estimation of remote processing times, given birth to a series of new techniques, which we will also denominate RTT (Remote Timing Techniques).

For a detailed description of timestamps, see RFC 1323.

The precision of the estimation depends on the frequency of the timestamp clock, which according to the RFC can vary from 1ms to 1sec. The implementation differences in the frequency of this clock are what render the method useful (or not) with respect to the techniques described here. In Appendix A you will find a table with the different timestamp clock frequencies by operating system.

Incidentally, timestamps can also be used in some cases to know the uptime of the remote system, and/or to distinguish between different systems in a load balanced environment. See Appendix B for references.

Timestamps are a relatively new addition to the TCP/IP protocols (1992, see RFC 1323), and are consequently not supported by all operating systems, are also not enabled by default in some OSs, and lastly, their implementation make timestamps not always useful with regard to the techniques described here.

TCP Intrinsic Operation

The other method found, which is in some cases much more "precise" (and therefore useful) than timestamps to discriminate between two different remote paths of execution, is related to the intrinsic operation of the TCP/IP protocol.

It is simply the fact that the ACK and "PUSH" flags can be sent together in the same packet, depending on if the application layer had passed the data or not to the TCP stack for delivery before the stack "ACKs" a previously received segment from the client. That is, if the server stack has something to ACK and at the same time has some data to send to the client, it "ACKs" and "PUSH" at the same time (in the same packet), but if the application is busy doing some other slow thing (like writing to disk) and didn't pass the data to the TCP stack quickly, the TCP stack sends a packet with only the pending ACKS (no data) and this difference can be easily detected on the client side. This technique is in some cases even better than timestamps, due to the fact that its "frequency" depends on the implementation of the TCP stack itself, and must be much faster (I don't know with certainty) than typical timestamp clocks. Moreover, this "frequency" is probably not fixed (i.e.: 100/sec) but depends on the speed of the processor, and its "precision" increases accordingly when the processing speed increases. The other advantage of this technique is that is common to all TCP implementations, and is therefore always "enabled" and available in all systems with a TCP stack.

Due that the timestamp information and the ACK and PUSH states depend completely on the operation of the remote system, and are passed to the client side as a kind of "snapshot" of the remote system, these methods are completely immune to network latency variations. They are also statistically immune to packet loss and load variations, and to other variable factors (disk IO, buffer cache, etc) being anyways necessary to increase the number of probes in case of small differences and/or great variations in the aforementioned factors.

Through careful examination of the behavior of the TCP data flow, observing the TCP headers at the right moments, is possible to statistically determine (by example) if a given username exists or not on a remote system, under the particular circumstances detailed below.

Proof of Concept

This example was tried successfully in two Linux boxes, one with Suse 7.0 (kernel 2.2.16) and the other with RedHat 6.2 with kernel 2.2.12. Both machines are relatively "old" machines, a Pentium II 366 and a Pentium MMX 233, both with standard IDE disks.

Many tests remain to be done, but I dare to say that the essential point seems to be mostly software related, that is, related to the way the remote application or service is written, than to the hardware used.

The techniques can be tried on any type of authentication process over TCP. Other uses of these techniques can also be found.

Linux has a timestamp clock frequency of 1000/s (1ms), which isn't very good, but is anyways useful. Delays greater than 1ms are typical of hard disks, and from the point of view of the application they remain statistically greater than 1ms independently of the buffer cache (at least with IDE disks).

Apropos, Cisco IOS implements a timestamp clock period of .1ms(!), although timestamps are disabled by default.

Description of Operation

Tcpdump is run in the background to capture the specific data flow (a telnet logon session) and an automated telnet session is initiated against the remote system, with a chosen username/password pair. The password is used as a mark in the stream.

The specific point at which the analysis must be made is detected (using the password mark), and the script outputs the TCP flags and the remote timestamp differences at that point. The later is possible due to the fact that Linux had timestamps enabled by default (2.2.x kernels) and also because it sends timestamps in both directions (encouraged by the RFC for simplicity).

If you note statistical differences between probes with a known to exist username (root, bin, lp, sys, etc) and probes with a known to be un-existent username (i.e.: dskhjgfjh), the remote system is vulnerable. The number of probes doesn't need to be very large (between 1 and 20) but in case of small differences, could be necessary to increase them. The probes will generate logs in the remote system (indeed, the processing time involved with these logs could be one of the causes of the timing differences) and, as long as you made more probes, more logs will be generated.

Statistical Differences Detected

- In this particular example, in case that the username exits on the remote system, the first packet sent by the remote system after the password was sent by the client, tends to be an “empty” (no data) ACK packet. This does not happen always, and some probes need to be done.
  Due to the fact that the data arrives almost always just a little bit later from the application layer to the TCP stack, and is then almost immediately sent to the client, the difference of the timestamps of the data packet and the previous ACK packet tends to zero in these cases.
- In case that the username does not exist, the packet tends to be a data packet (ACK+”PUSH”). The timestamp differences with the next packet (typically other ACK+data packet, the next “login:” banner) tend to be relatively big.

So, when the username exists, the mean of the timestamp differences of the probes tend to be smaller than the mean of the differences if the username does not exist.

If timestamps are not enabled, you can detect the difference anyway (!): The number of times you’ll see an “empty” (no application data) ACK packet (again at the right place of the data stream) will be greater if the username exist than the number of times you’ll see it if the username doesn’t exist.

Of course, even without this information, the difference could be statistically detected anyway. More (maybe many more) probes will be necessary, to overcome the measurement errors introduced mainly by network latency variations. On networks with high latency variations, this could yield the “attack” highly unpractical.

The techniques described here are useful to diminish the measurement errors over TCP network links, diminishing then the number of probes or samples necessary to detect a timing difference, thus yielding some otherwise unfeasible or expensive attacks feasible.

Other Services

Other tested services known to be vulnerable to this kind of “attack” are:

- rlogin: rlogind also gives us the same information, but after two passwords are entered. (In the pause between the second password entry and the “Login incorrect” message.
  As long as the standard rlogin command “clears” its standard input before asking for a password, no automation is possible without replacing it with other version or reproducing the protocol manually.
- ftp: wu-ftp seems to be vulnerable also, just after entering the username (after the USER command). A slight difference of around 1ms was statistically detected using the timestamp information, on the Suse distribution.

Services that remain to be tested:

- imap
- pop
- rexec
- Auth services over http.
- ssh: ssh (or any other encrypted protocol) is a particular case, due to the fact that an eavesdropping attack makes sense in these cases. The techniques described here could be used (by example) to extend Paul Kocher attack (see References) over a network protocol (i.e. to make the attack remotely possible).
- …

All tests were done only on Linux, mainly on the machine with the Suse distribution.
Other operating systems and services remain to be tested.

Probable cause of the differences

The probable cause (not verified) of the differences in both systems seems to be the difference in the logging done trough syslog if the username exists or if not. In the Suse distribution, particularly, faillog logging is enabled by default, and this logging occurs only if the username exists (and the logon failed, of course).

Other probable (also unverified) cause of the differences could lie inside the implementation of the various pam modules on which Linux authentication depends. If pam were the cause, the same effect would have to be observable in all the services that relay on pam for authentication (the default).

Possible Solutions

As long as at least one of the techniques depends on the intrinsic operation of the TCP stacks, no easy solution seems to be available.

With regard to timestamps, a partial solution would be to increase the period of the timestamp clock, let's say, to 50ms or more. But probably, statistical differences would show up in the long term.

The "problem" of the intrinsic operation of the TCP stack would still remain.

The only definitive solution seems to be to modify the code of the vulnerable services, in order to avoid the "leaking" of timing information, by example introducing random delays, or better, always sending data packets to the client with no pending ACKS in them (and random delays for timestamps). A "send" function with these characteristics wouldn't be so difficult to write and use.

Other Applications And Uses

I'll quote here a comment by Paul Kocher, who told me in a private communication

```
"You might want to try some … statistical attacks …
… -- using them, even very tiny differences (<1 us) can
be resolved even if there is quite a lot of measurement error
(>1 ms)… . The general math required
is quite simple - you'd want to look for the difference between
the *average* time when [for example] n bytes of a password
are correct and the average time when n+1 bytes of the password
are correct."
```

That is, the only necessary thing to detect a timing difference is to take enough samples or to make enough probes for the difference to become statistically appreciable. The number of probes or samples will be directly proportional to the measurement error (that is, to the variance of the measurements), and inversely proportional to the variance of the timing difference to detect (from Kocher paper).

I want to make clear here, before generating unnecessary alarms, that Mr. Kocher is talking about an hypothetical case. At least in Unix, passwords can't be revealed this way due to the fact that in the encryption process the correlation between clear text and encrypted password is lost; an incorrect clear text password "closer" to a correct one, does not yield an encrypted password closer to the real encrypted password. Thus, no "password approximation" is possible.

An obvious practical use of these techniques is as an addition to, for example, a brute force username/password tester, to avoid trying username/password combinations with an inexistent username. The program could start testing username/password pairs normally, at the same time that it monitors the data flow to statistically detect the actual username existence or inexistence, changing its behavior accordingly.

Timestamps could also be used to precisely measure inter keystroke timings of an interactive encrypted session; so, they can be used, by example, to greatly improve the efficiency and precision of the "Herbivore" system (see Wagner et al), especially over networks links with big latency variations.

Other uses of these techniques could be discovered or developed. (directory and/or file discovery?, fingerprinting issues?, …?). Generally speaking, whenever a timing difference is identified(by source code analysis, by example), RTT could be used to "measure" or detect that difference remotely over TCP.

Source code

The appendix B contains scripts that will do the hard work for you. You probably would have to make some manual adjustments, mainly the interface name(s), needed by tcpdump.

The scripts were written without taking into account security practices (i.e. writing "secure" code). They are probably even remotely exploitable.

Conclusions

Although the particular example shown here does not yield a great deal of information (just the knowledge of if a given user exists or not on a remote system) and is successful only if a series of conditions are met, the techniques used to know this little bit of information are new (as far as I know), and probably they could be used in many other ways.

TCP timestamps could yield a “precise” timing of the remote processing paths, what can be potentially useful for many different purposes. In particular, some operating systems seem to have a much more precise timestamp clock than Linux, and this open new possibilities to experimentation. On the other side, given the fact that one of the techniques (observing TCP flags) is intrinsic to TCP operation, its application is feasible wherever a TCP connection is being used.

Acknowledgments

Thanks to David Wagner, Paul Kocher and Brett McDanel for their suggestions and comments.

Appendix A: Table of Operating Systems and Timestamp Clock Frequency.

See Bret McDanel Work (Appendix B)

Appendix B: References and Related Works

RFC 1323	http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1323.html
Bret McDanel Paper on Timestamps	http://www.mcdanel.com/bret/research/tcp_timestamp.jsp
Paul Kocher Timing Attack Paper	http://www.cryptography.com/timingattack/paper.html
Dawn Xiaodong Song, David Wagner and Xuqing Tian Paper on Timing Attacks on SSH	http://www.usenix.org/publications/library/proceedings/sec01/song.html
PDF Version of this Paper	http://www.maurol.com.ar/security/RTT.pdf

Appendix C: Source Code

```
------------------------------------------------------------rtt.sh---------------------------------------------------------------------
#!/bin/bash
# Probe a telnet service using remote timing techniques
# to determine if a user exist or not.
# Proof of Concept code.
# (C) 2002 maurol (maurol@mail.com)

# Customize these if needed:
LOOPBACK=lo
LAN=eth0
#WAN=eth0
WAN=ppp0

# Default interface to listen to
#IFACE=$LAN
IFACE=$WAN

# Default port
PORT=23
# Default number of probes
COUNT=10
# Time in seconds to wait before sending the username. For slow links
# and/or no reverse dns resolution this could be as big as 25.
LAN_SLEEP=1
WAN_SLEEP=6
#SLEEP=$LAN_SLEEP
SLEEP=$WAN_SLEEP
```

```
if [ $# -lt 2 ]
then
 echo "Usage: $0 <ip> <user> [count] [sleep]"
 echo "ip   : IP address of remote system."
 echo "user : Username to probe."
 echo "count: Number of probes."
 echo "sleep: Initial sleep in seconds."
 exit 1
fi

HOST=$1
USER=$2

[ "$HOST" = "127.0.0.1" ] && IFACE=$LOOPBACK
if echo $HOST | grep -E "^10\.|^192\.168\." >/dev/null
then
 IFACE=$LAN
fi

[ "$IFACE" = "$LOOPBACK" ] && SLEEP=$LAN_SLEEP
[ "$IFACE" = "$LAN" ] && SLEEP=$LAN_SLEEP
[ "$IFACE" = "$WAN" ] && SLEEP=$WAN_SLEEP

PASS=zzzzz
export PASS

[ -z "$3" ] || COUNT=$3
[ -z "$4" ] || SLEEP=$4

export IFACE HOST PORT USER

MEAN=./mean
VAR=./var

[ -x $MEAN ] || cc -o mean mean.c
#[ -x $VAR ]  || cc -o var -lm var.c

>tests.$USER
c=0
while [ $c -lt $COUNT ]
do
# Launch the capture script in the background
 (./capture.sh >/tmp/regs.$$ ; ./dif.sh /tmp/regs.$$ | tee -a tests.$USER; rm -f
/tmp/regs.$$) &
# Start telnet session
 (sleep $SLEEP; echo $USER; sleep 1 ; echo $PASS ; sleep 3)| telnet $HOST
 c=`expr $c + 1`
 sleep 1
done
echo
(
cat tests.$USER | egrep -v "^[      ]*$"
echo -n "mean:    " ; cat tests.$USER | egrep -v "^[  ]*$" | $MEAN 2 ;echo
#echo -n "var.:    " ; cat tests.$USER | egrep -v "^[  ]*$" | ./var.sh 2 ;echo
) | tee stats.$USER ; rm -f tests.$USER
```
--capture.sh--

```
#!/bin/bash
# Captures the telnet session and seeks the useful timestamps and flags.

HEX=/usr/bin/hex
TCPDUMP=/usr/sbin/tcpdump

if [ -x $HEX ]
then
 PATTERN=`echo $PASS | $HEX -w 3 -g | head -1 | sed "s/^[^ ]* //;s/ //" | cut -c-
7| sed "s/[     ]*$//"`
else
 PATTERN="7a7a 7a"       # "zz z" ;-)
fi

if [ -x $TCPDUMP ]
then
 $TCPDUMP -l -n -n -i $IFACE -x port $PORT > /tmp/lst &
else
 echo "$TCPDUMP not found or not executable!"
 exit 1
fi

sleep 2
while ! grep "$PATTERN" /tmp/lst >/dev/null
do
 sleep 1
done
sleep 3
sed -n "/$PATTERN/,\$p" /tmp/lst | grep "$HOST.$PORT >" | sort -u | head -2 | sed
"s/^.*$HOST.$PORT >//" | awk '{print $2" "$8" "$9}'

killall tcpdump
rm -f /tmp/lst
```
--dif.sh--
```
#!/bin/bash

FLG1=`head -1 $1 |awk '{print $1}'`
TS1=`head -1 $1 | grep timestamp | awk '{print $3}'`

FLG2=`head -2 $1 |tail -1 |awk '{print $1}'`
TS2=`head -2 $1 |tail -1 | grep timestamp | awk '{print $3}'`

DIF1="-"
[ -z "$TS1" ] || [ -z "$TS2" ] || DIF1=`expr $TS2 - $TS1`

echo "$FLG1 $DIF1"
```
--mean.c--
```
#include <stdio.h>
#include <stdlib.h>

#define MAXLINE 1024
#define MAXVAL  MAXLINE

main(int argc, char **argv)
{
 int col=1,c=0,v=0;
 float s=0;
```

```
        float va;
        char *line,*pline, *val;
        line=(char *)malloc(MAXLINE);
        val =(char *)malloc(MAXVAL);
        pline=line;

        if (argc>1)
                col=atoi(argv[1]);

        while (line=fgets(line, MAXLINE, stdin)) {
                for(v=0;v<col;v++)
                        val=strsep(&line, "     ");
                line=pline;
                if (val[0] == '\0')
                        continue;
                va=strtod(val,NULL);
                s=s+va;
                c++;
        }
        printf("%.6f      ", s/c);
       }
```

Source Code Download http://www.maurol.com.ar/security/rtt.tgz

Author

Mauro Lacy (maurol@mail.com)