

# TCP timestamp & advanced fingerprinting

**Source:** <http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2005-03/0450.html>

---

**From:** Erwan Arzur ([erwan\\_at\\_lse.epita.fr](mailto:erwan_at_lse.epita.fr))

**Date:** 03/25/05

Date: Fri, 25 Mar 2005 15:04:42 +0100

To: [bugtraq@securityfocus.com](mailto:bugtraq@securityfocus.com)

-----BEGIN PGP SIGNED MESSAGE-----

Hash: SHA1

Hello,

attached is a paper from one of our students about using the TCP timestamps in TCP headers as a fingerprinting tip, which can ultimately be used for mapping networks behind firewalls.

Erwan Arzur

EPITA/EPITECH systems Laboratory

<http://www.lse.epita.fr/>

-----BEGIN PGP SIGNATURE-----

Version: GnuPG v1.4.0 (FreeBSD)

Comment: Using GnuPG with Thunderbird - <http://enigmail.mozdev.org>

iD8DBQFCRBp5tchshDF9KNYRApEaAJ9Cx+AfvdFyHMl14nBo3ZCrPRzK8ACfasSP

Wkx5F+xTG9+BGD/wmWFeOBM=

=2T62

-----END PGP SIGNATURE-----

-----BEGIN PGP SIGNED MESSAGE-----

Hash: SHA1

Advisory - March 19th 2005

Advanced fingerprinting on TCP timestamping - defeat the IP masquerade !

Original advisory on <http://www.lse.epita.fr/publications.php>

---

Synopsis :

-----

## SecurityFocus Bugtraq: TCP timestamp & advanced fingerprinting

This network oriented advisory explains how to fingerprint network services on a remote host. After reading this paper you will be able to circumvent the illusion given by IP maquerading that network services are hosted by a single computer.

Intro :

-----

While working on this technique I found some papers related to TCP timestamping and applicable to fingerprinting, but none of these papers I read has covered the issue developed in the current advisory.

[Note added on March 24th 2005 : a whitepaper about "physical device fingerprinting" has been released in March during the time I was working on the current issue. You may find it at <http://www.caida.org/outreach/papers/2005/fingerprinting/>]

If you know tools applying such methods, feel free to contact me. Either way, I made a small scanner which acts as a proof-of-concept tool. You may find it on <http://www.lse.epita.fr/developpements.php> or at <http://www.frhack.org/masqb/>.

There is a well-known fingerprinting method which is based on TCP timestamps, allowing to detect for how long a computer has been up. This method was first used by the website netcraft.com and is now a common feature of tools like nmap. We will first explain the history of this method.

Then, when you understand how TCP timestamping works, a detailed analysis of various operating systems should reveal how analyzing timestamps further can be used in breaking IP masquerades.

TCP Timestamping :

-----

On March 11st 2001, Bret McDanel published an introduction on the security mailing list Bugtraq describing the feature of the netcraft.com website to offer it's visitors an overview about the current and average uptime of specific operating systems running on the machines that provide the base for popular websites.

TCP timestamping (optional TCP field) is explained in RFC (Request For Comments) 1323 and is a method used notably in PAWS (Protection Against Wrapped Sequence Numbers).

But we will just keep in mind the elements brought to us by McDanel : the TCP timestamping is not of a security concern, however the way operating systems manage their TCP timestamping is "interesting" : it allows a remote client to guess, if he recognizes the operating system (OS fingerprinting), the time the machine is up.

When you start up your BSD, by example, the operating system increments, as explained by McDanel, the timestamp value of one point each 500 milliseconds. By grabbing the value of timestamp of such operating system you can guess for how long it is still running.

## SecurityFocus Bugtraq: TCP timestamp & advanced fingerprinting

The RFC does not give many indications on how timestamps should be managed inside operating systems, only that it should increase according to the fact that "it must be at least approximately proportional to real time".

Here is the format of this optional TCP field :

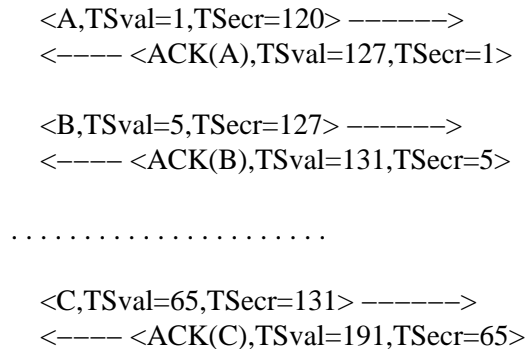


There are a few limitations that appear in the prediction of the actual uptime.

First the length of a timestamp value in a TCP packet is 4 bytes, so it will roll over when the value crosses the limit of  $2^{32}$ . Additionally, some operating systems (Windows being the example from McDanel), may not instantly start to increase their timestamp once the system has been booted up.

TCP timestamps values are inserted in many TCP packets (such as SYN, ACK...) but timestamp replies are mainly part of ACK packets. RFC 1323 gives the following schema :

TCP A TCP B



The TCP stacks of Linux kernels (2.4 and 2.6) natively set a timestamp value in their SYN packets, but Windows-based operating systems does not.

The TCP stacks of Linux kernels (2.4 and 2.6, again...) natively set a timestamp value in SYN/ACK packets, but again Windows does not.

So, now you should be able to understand that it is important to analyze how various TCP stacks react with setting their timestamps inside the packets if we want to take advantage of these differences.

Breaking the masquerade :

-----

The first thing I did was performing an experiment with a simple sniffer and common SYN packets sent from a Linux box. This test was intended to report

## SecurityFocus Bugtraq: TCP timestamp & advanced fingerprinting

if a specific operating system replies a timestamp value (TSval) in a SYN/ACK packet and if the incrementation is predictable (you may predict on two successive connections that next value in second connection will be higher than in first).

Operating system	answers a TSval*	use TSval	predictible TSval
Linux 2.6/grsec	Yes	Yes	Yes
Linux 2.6	Yes	Yes	Yes
OpenBSD 3.6	Yes	Yes	Yes
FreeBSD 5.3	Yes	Yes	Yes
NetBSD 2.0	No	Yes	No
Windows 2000/SP4	No	Yes	Yes
Windows XP/SP2	No	Yes	Yes
Windows 2003	No	Yes	Yes
MacOS X	Yes	Yes	Yes
Solaris 9	Yes	Yes	Yes
OSF1 4.0	No	No	-

\*: in SYN/ACK packet

It can be noted that some systems are less verbose than other during the three way handshake connection.

The sidenote that the timestamp is predictable (or quite predictable at least) bears a subtle reason : if the timestamp were to be defined randomly or zeroed for each new connection, it would be quite impossible to compare it to existing data we already collected.

So, what is this all about now ? First, the implementations of packets carrying timestamps and other packets not carrying timestamps in the different operating systems allows for a possible distinction of the operating systems that have generated those packets.

Second, it would be possible to differentiate between machines where the packets are coming from by looking at and comparing separate timestamp values. And last but not least, looking at TTL values should also reveal the relative "distance" in hops to the machine providing the services you are looking at.

That means, for example, that you may be able to detect whether a given web site runs on only one web server or multiple machines behind some kind

## SecurityFocus Bugtraq: TCP timestamp & advanced fingerprinting

of firewall or load balancer, sitting in a local LAN and being presented with only one global IP to the WAN.

Another strong advantage of the proposed method is that you can enumerate the machines behind an IP masquerade, and, with timestamp analysis, link together the web services with the corresponding system. It is the method implemented in the proof-of-concept software I designed.

```
$ ./masqb 10.42.42.42 21 22 80 443
(...)
Results for 10.42.42.42 :
[+] System A listens the following ports : 21
[+] System B listens the following ports : 22
[+] System C listens the following ports : 80 443
Game over.
```

As Windows sends timestamp values in each established connections, only NetBSD and OSF1 are not covered by this problem.

NetBSD 2.0 initializes timestamp at 0 in each new connection, OSF1 does nothing.

If services are forwarded on both a Linux system and a NetBSD, you should be able to easily detect it.

If services are forwarded from a set of homogeneous NetBSD or OSF1 systems, then you will hardly obtain any valueable results.

Finishing this advisory, I can conclude that this technique applies to the magnitude of systems available on the public Internet.

The only thing you need is at least two open ports with services accepting and serving incoming connections (one port for load balancing detection).

It has already been successfully used for retrieving information about network physical infrastructures and network services behind firewalls and various forms of routers.

Solution :

-----

Can your firewall defuse the timestamp ? Else take a look at :

```
$ sysctl -a | grep tcp_timestamps
```

--- Author : Clad Strife.

LSE - Epita/Epitech System Laboratory

++ Thanks to : Alexander Gabert (grammar expert and much more ;-)), gumleef (also an expert of something, don't ask me what...), Hal9000 (sysctl expert).

-----BEGIN PGP SIGNATURE-----

Version: GnuPG v1.0.6 (GNU/Linux)

Comment: For info see <http://www.gnupg.org>

SecurityFocus Bugtraq: TCP timestamp & advanced fingerprinting

iD8DBQFCRBRDKvqs9JUR15URAg11AJ9mQx+34KtmFuSf2IhdJeOr9PHUogCeP5ns  
NWfErlqZvKI+50HYVRpRfFU=  
=3Lgk  
-----END PGP SIGNATURE-----