

TCP/IP interception

Or, “You want to do what with my data?”

Adrian Chadd <adrian@FreeBSD.org>

Why?

- Lots of “interception” is going on as part of service delivery
- Is “good enough” but most get it subtly wrong
 - (And then blame Squid)
- I’d like FreeBSD to get it “right”

Hijacking 101

- Lots of places to “hijack” traffic
- The interesting bits - flow interception, local interception
- Two major kinds of networks
 - service provider
 - content provider

CP Hijacking

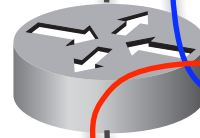
1. Client makes request.



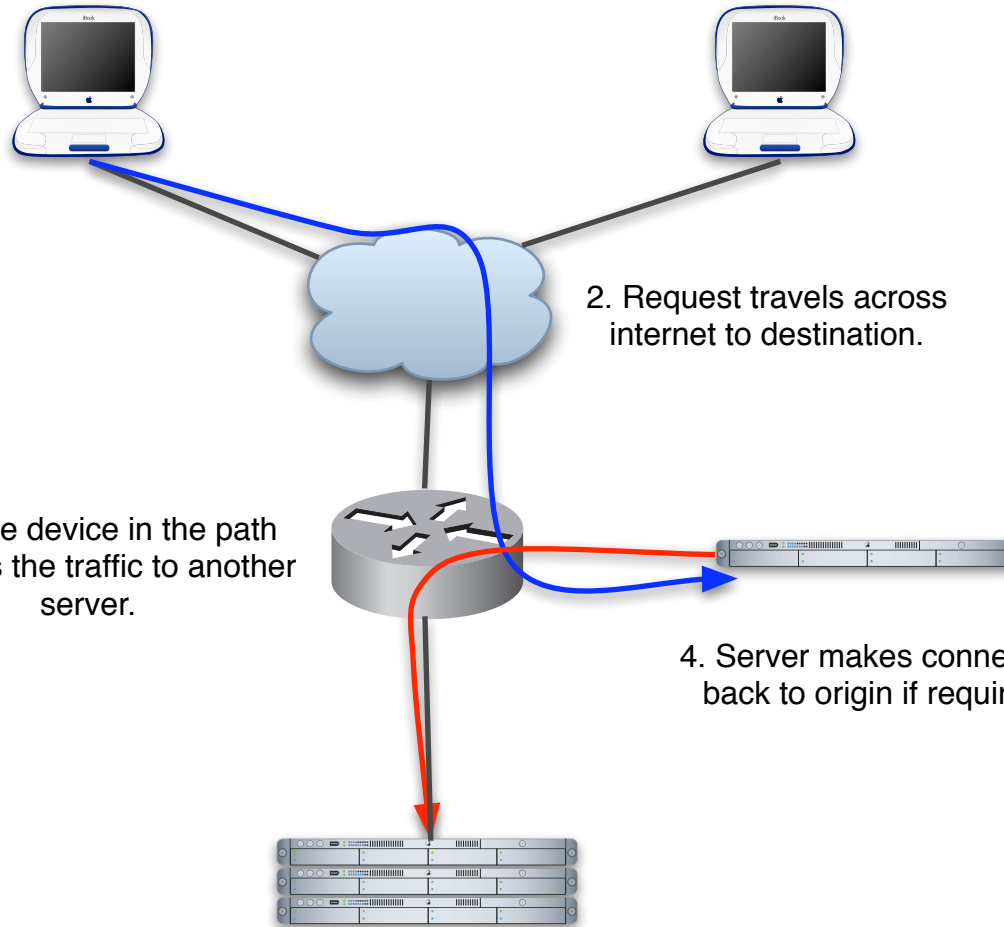
2. Request travels across internet to destination.



3. Some device in the path redirects the traffic to another server.

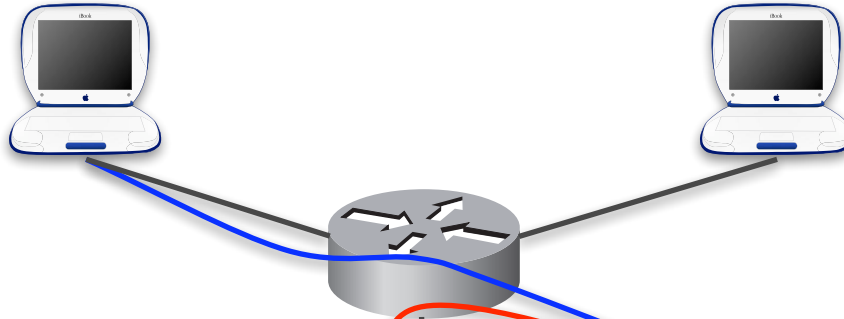


4. Server makes connection back to origin if required.

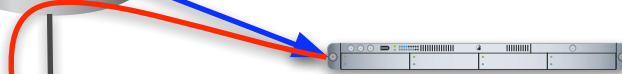


SP Hijacking

1. Client makes request.



2. Some device in the path redirects the traffic to another server.



3. Server makes connection back to origin if required.



4. Request travels across internet to destination.



Differences?

- Client location
 - SP = local(ish) clients; CP = remote(ish) clients
- Services location
 - SP = remote(ish) servers; CP = local(ish) servers
- Network topology
 - Asymmetric vs Symmetric?

Practical Hijacking

- Get packet stream to server somehow!
 - Inline; SLB; L{>=3} switch; policy routing; ECMP; WCCPv2
- Server intercepts TCP/IP connection not destined for itself (ipfw fwd, etc); creates PCB, etc;
- Software treats it as a normal incoming TCP/IP with a strangely remote “local” socket name

Problems with Hijacking

- All the UNIXes ship with enough to do server-side hijacking (ie, incoming connections)
- Outgoing connections still look like they originate via the server, not the original client
- Everyone gets TCP options, PMTU and MSS all wrong; ECN isn't so bad these days
- The workarounds are annoying!

Hijacking: ECN

- ECN has been around a long time; reused bits in header
- Linux had ECN support for quite a while
- Clients didn't!
- Some firewalls treated ECN as “should be 0” and dropped ECN packets on the floor
- Clients could talk fine; intercepted connections via Linux proxies couldn't..
- **Workaround:** disable ECN.

Hijacking: MSS

- Negotiated MSS on outgoing connections is not the same as the incoming connection
- Which is not a big deal in itself
- Many CPE's "negotiate down" the MSS to bypass tunnelling MTU related issues (PPPoE, bad MPLS, IP/GRE, IPSEC)
- .. and this leads to..

Hijacking: PMTU

- .. PMTU requires ICMP to function correctly
- Hijacking connections generally involves intercepting TCP and **NOT** ICMP
- .. so all kinds of hilarity follows
 - not specific to intercepted client traffic!
- **Workaround**: disable PMTU on server; clamp connection MSS to something low; pray.

Hijacking: Options

- Client-side options differ from FreeBSD-side options
 - Especially in FreeBSD-7.0
- Specific option in question: TCP MSS
- Some routers/etc just zero the TCP MSS field
- Both sides of the connection mis-negotiate TCP MSS; hilarity follows
- **Workaround:** disable TCP MSS, cry.

Client-side spoofing?

- TPROXY (linux)
- Julian's patch set (in perforce somewhere)
- Allow for outgoing TCP/IP connections with a non-local "local" bind()
- Things which assume end-to-end stay happy
- Client and server have no idea there's something in the middle..
 - .. or do they?

Client-side spoofing

- How it works:
 - (compile kernel with stuff)
 - `fd = socket(normal stuff);`
 - ipfw rule to push non-local packets through a PCB lookup (eg, ipfw uid..)
 - `setsockopt(fd, IPPROTO_IP, IP_NONLOCALBIND, "yes");`
 - `bind(fd, "non local address") = OK`

Problems!

- It assumes symmetric traffic flows
- Similar to traditional interception - PMTU, MSS, Options
- Other things may differ - SACK/ Timestamps, ensuring source/dest ports match (TPROXY + Squid doesn't do this!)

Current Plans

- Document the “how hijacking is done wrong” cases somewhere public
- Integrate Julian’s client spoofing patches into -current
- Release Squid as an example use case (already has the logic for TPROXY)
- Tidy up my “TCP proxy” as a simple example for interception testing
- Investigate TCP/IP “de-splicing”

TCP/IP “desplicing” ?

- Track TCP FSM on packets passing by
- Wait until an entire connection setup is seen that you care about
- Create two sockets - client and server - setup relevant PCBs; return to userland
- Bypasses the “symmetry” issue - asymmetric paths are never intercepted
 - Just count the sessions you couldn't hijack

TCP/IP desplicing issues

- Problems (that I can think of)
 - Unknown IP/TCP Options?
 - Unused server-side connections? (eg caching)

Questions?

Thankyou!

Adrian Chadd <adrian@FreeBSD.org>