
BLACK OPS OF
TCP/IP
Spliced NAT2NAT And Other
Packet-Level Misadventures
Dan Kaminsky, CISSP
DoxPara Research
www.doxpara.com

Where I'm Coming From…

- Black Hat / DefCon 0x7D1
  - Impossible Tunnels through Improbable Networks with OpenSSH
    - *Getting Out:*
      ProxyCommands for Non-TCP comm layers
      - HTTP, SOCKS, UDP, Packet Radio*, AIM/Yahoo*
    - *Coming In:*
      Active Connection Brokering for NAT2NAT
      - One host **exports** SSHD **to** broker
      - Other host **imports** access **from** broker
    - *Passing Through:*
      Dynamic Forwarding for Psuedo-VPN Work
      - Web Browsing, Dialpad(Split-H323), etc.

Interesting Problems

- Instant Portscan
  - “Is it **possible** to discover instantaneously what network services have been made available, even on massive networks?”
- Guerrila Multicast
  - “Is it **possible** to send a single packet to multiple recipients, using today’s multicast-free Internet?”
- “NATless NAT”
  - “Is it **possible** to share a globally addressable IP address without translating private IP ranges a la NAT?”
  - Is it **possible** to allow incoming connections to an IP multiplexed in this manner?
- NAT Deadlock Resolution
  - “Is it **possible** to establish a TCP connection between two hosts, both behind NATs?”

On Possibility

- Restraint Free Engineering
  - “Abandon All Practicality, Ye Who Enter Here”
  - “It’s amazing what you can do once security is no longer a concern.”
- You’ve got what you’ve got. Make interesting things happen.
  - It might end up practical.
  - It might end up secure.
  - Right now, it’s impossible. **Fix that first.**
    - Maybe.

On Packet Structure

- Packets are “strangely ordered”
  - Where it’s ending up next, where it came from recently, how it’s hopping from one place to the next, how it’s hopping to its final destination, checksum, where the packet came from originally, where it’s going to end up, what app it came from, what app it’s going to, checksum, god knows what, ANOTHER checksum
- Why not sort everything; put all the “came from” and “going to’s” Why so much redundancy? Isn’t it inefficient?
- WHO CARES?

Layers: Not What, But Who

- One medium, many messages
  - Listeners reconstruct meanings relevant to themselves, ignore the rest
  - Managed (ir)responsibility
- Fields are out of order, occasionally because they're addressed to different entities
  - Name and address repeated inside a business letter and on the envelope
- **Messages at one layer can modulate messages received at another**
  - Insufficient postage will prevent a correctly addressed letter from getting sent
  - Incorrect internal address has unknown effects

Error Recovery Per Layer

- Layer 2 (Point to Point)
  - Errors are quickly recoverable, but error generation can occur at same layer as layer control
    - Data is destroyed and recreated each frame
  - Corporate Fertilizer
- Layer 3 (Router to Router)
  - Many more sources of personally irrelevant error
  - Highest Traffic Link
  - Data is modulated – minimum change possible
- Layer 4 (End to End)
  - Lowest traffic, highest personal relevance
  - Errors here actually matter

- This slide intentionally left blank

TCP Connection Traits: Flags

- Connection Request (Alice -> Bob)
  - SYN: I want to talk to you
- Connection Response (Bob -> Alice)
  - SYN|ACK: OK, lets talk.
  - RST|ACK: I ain't listening
- Connection Initiation (Alice -> Bob)
  - ACK: OK, beginning conversation.

TCP (and UDP) Connection Traits: Ports

- Local Port: What application requested the connection. Usually a random number, 0-65535.
  - 0 is a valid port
- Remote Port: What application accepted the connection. Usually a “known number”
  - 80 for HTTP
  - 143 for IMAP
  - 443 for HTTP/SSL
- IP handles who we’re talking to; Ports handle what we want from them

TCP Connection Traits: Sequences

- Sequence Numbers
  - 32 bit number, randomly generated, must be reflected by the opposite party in a TCP handshake
  - After initial reflection, used to relay information about successful packet acquisition

Connection Summary

- Flag determines phase
  - Asymmetric
- Port determines process
- Sequence “secures session”
  - Prevents trivial spoofing attacks
  - Also used to manage connection speed, identify which bytes are being acknowledged

Stateless Pulse Scanning

- Instant Portscan
  - “Is it **possible** to discover instantaneously what network services have been made available, even on massive networks?”
- Answer: Yes, practically, even securely
  - Separate scanner and listener processes
  - Sending
    - Directly send n SYN packets
    - Same local port
    - SYN cookies
  - Receiving
    - Kernel filter packets arriving to local port
    - Record connection phase: Port up(SYN|ACK) or host up, port down(RST|ACK)

Issue: Spoofed Responses

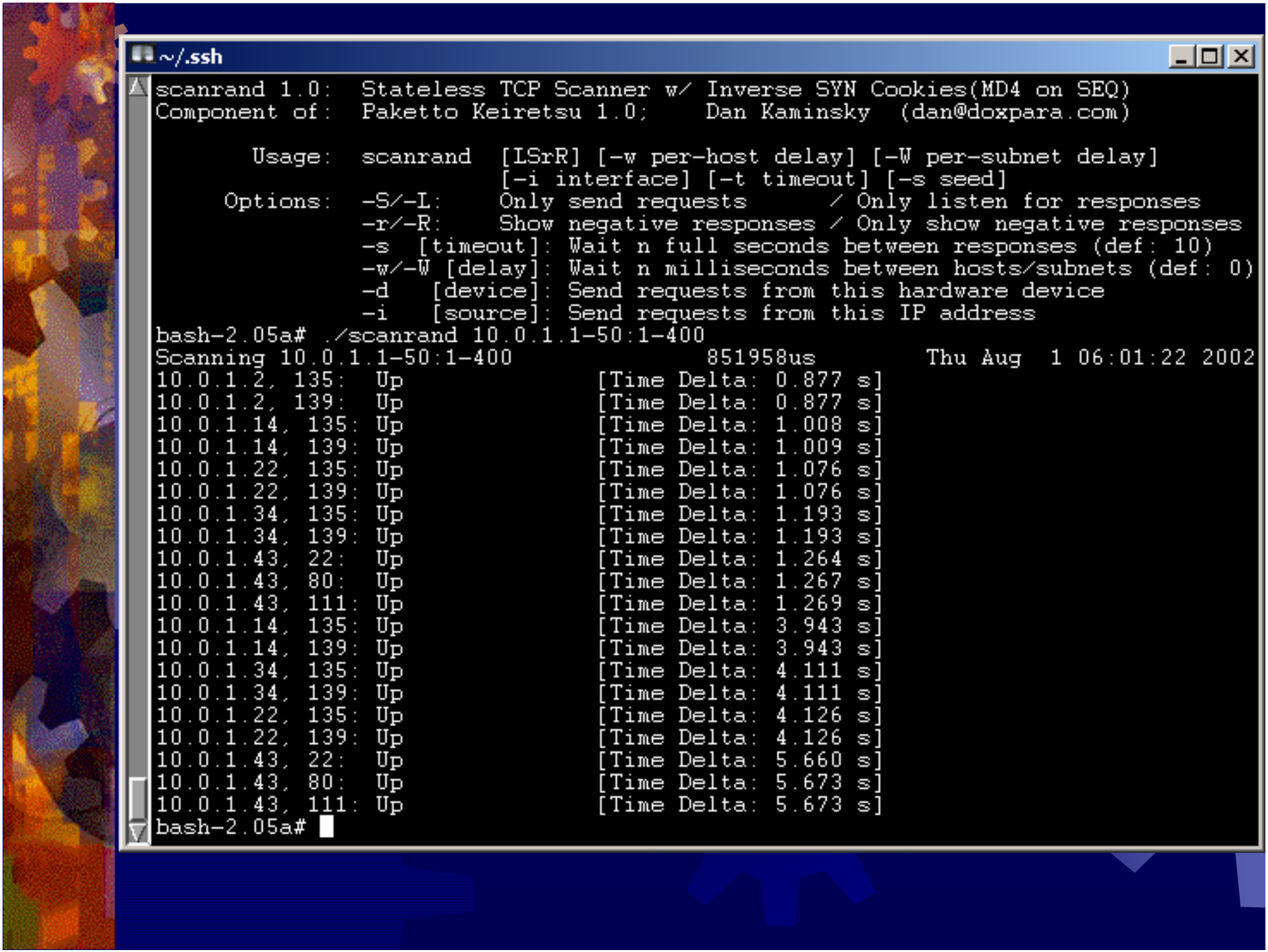
- Easy to spoof hosts being up if the scanner isn't tracking who (or how it scanned)
- Solution: Invert SYN Cookies!

SYN Cookies

- Developed in ’96, when SYN floods became common
  - ACK reflects ACK# of SYN|ACK(incremented by one)
  - Encrypts connection state into the SYN|ACK’s ACK#
  - Therefore, you can use legitimate remote hosts – instead of kernel memory – to store handshake state
- Ahhh…but SYN|ACK also reflects SEQ# of SYN in its ACK#…
  - Instead of tracking SYN|ACK reflections in the ACK, track SYN reflections in the SYN|ACK

Implementation: Scanrand 1.0

- **Element of: Paketto Keiretsu**
- 384 lines of libnet and libpcap, w/ trivial MD4 include
  - **No state stored**
  - **Scans at ~11-20mbit**
  - **Possibly even portable**
- **100% complete, release imminent**


~/.ssh
scanrand 1.0: Stateless TCP Scanner w/ Inverse SYN Cookies(MD4 on SEQ)
Component of: Paketto Keiretsu 1.0; Dan Kaminsky (dan@doxpara.com)
Usage: scanrand [LSrR] [-w per-host delay] [-W per-subnet delay]
[-i interface] [-t timeout] [-s seed]
Options: -S/-L: Only send requests / Only listen for responses
-r/-R: Show negative responses / Only show negative responses
-s [timeout]: Wait n full seconds between responses (def: 10)
-w/-W [delay]: Wait n milliseconds between hosts/subnets (def: 0)
-d [device]: Send requests from this hardware device
-i [source]: Send requests from this IP address
bash-2.05a# ./scanrand 10.0.1.1-50:1-400
Scanning 10.0.1.1-50:1-400 851958us Thu Aug 1 06:01:22 2002
10.0.1.2, 135: Up [Time Delta: 0.877 s]
10.0.1.2, 139: Up [Time Delta: 0.877 s]
10.0.1.14, 135: Up [Time Delta: 1.008 s]
10.0.1.14, 139: Up [Time Delta: 1.009 s]
10.0.1.22, 135: Up [Time Delta: 1.076 s]
10.0.1.22, 139: Up [Time Delta: 1.076 s]
10.0.1.34, 135: Up [Time Delta: 1.193 s]
10.0.1.34, 139: Up [Time Delta: 1.193 s]
10.0.1.43, 22: Up [Time Delta: 1.264 s]
10.0.1.43, 80: Up [Time Delta: 1.267 s]
10.0.1.43, 111: Up [Time Delta: 1.269 s]
10.0.1.14, 135: Up [Time Delta: 3.943 s]
10.0.1.14, 139: Up [Time Delta: 3.943 s]
10.0.1.34, 135: Up [Time Delta: 4.111 s]
10.0.1.34, 139: Up [Time Delta: 4.111 s]
10.0.1.22, 135: Up [Time Delta: 4.126 s]
10.0.1.22, 139: Up [Time Delta: 4.126 s]
10.0.1.43, 22: Up [Time Delta: 5.660 s]
10.0.1.43, 80: Up [Time Delta: 5.673 s]
10.0.1.43, 111: Up [Time Delta: 5.673 s]
bash-2.05a#

Observed Results

- Since no state is maintained within the scanner, we can send SYNs at wire speed
  - Implementation can get faster
- Found ~8300 web servers on a corporation's Class B
  - Time spent: ~4 Seconds
- Collisions
  - Initial SYNs might collide, but SYN|ACKs **resend**
- SYN|ACKs are given RSTs by present kernels automatically
  - The SYNs were generated in userspace – the kernel has no idea the connection request was ever sent

Implications

- Userspace manipulation of packets can lead to less overhead
  - Kernels are optimized to talk to other hosts, not simply to scan them
- Packet content can be overloaded
  - A random field can always be replaced with encrypted data (and vice versa)
    - This is the heart of kleptography
- Elegant solutions sometimes can be reapplied elsewhere
  - SYN(really SYN|ACK) cookies made SYN reception more efficient
  - Inverse SYN cookies make SYN transmission more efficient

Layer Redundancy

- L2: Broadcast MAC Address
  - FF:FF:FF:FF:FF:FF
  - Absolute
- L3: Broadcast IP Address
  - Last IP of Subnet
  - Relative
  - Sending to it is known as a Directed Broadcast
    - Often blocked, if it can be detected
    - Detection can be…suppressed.

Broadcast GHosts

- Guerrila Multicast
  - “Is it **possible** to send a single packet to multiple recipients, using today’s multicast-free Internet?”
  - Answer: Yes, barely.
- Link a unicast IP to a broadcast MAC address; all responses to that IP will be broadcast throughout a subnet
  - No individual client need duplicate the datastream – the switch will issue copies of the data to all downstream hosts

IP Incorporated

- DHCP for an IP
  - May or may not use broadcast MAC in DHCP request – just trying to validate that nobody else is using the IP
- Answer ARP requests for that IP with Broadcast MAC (or Multicast MAC)
  - At L2, w/o IGMP Snooping working, Multicast = Broadcast
- Issue L4 requests against a remote host, unicasted via layer 3, with responses broadcasted locally at layer 2
  - Elegance has left the building

Firewall Issues

- NAT
  - 100% NAT penetration, as long as the implementation doesn't refuse to NAT for a broadcast MAC
    - PIX, which accepts…Multicast MACs!
  - Multicast through NAT!
- UDP
  - Remote side can send data forever – as long as it keeps packets coming in before the UDP state expires, no further data is required from behind the wall

TCP w/ Guerrila Multicast

- Without any listeners, stream dies
- With one listener, stream can operate normally
- With many listeners, only one should participate in acknowledging the stream
  - If that one dies, another should take its place
- Solution: Random delays
  - On reception of a packet to be acknowledged, queue a response within the next 50-1500ms
  - Broadcast response
  - If another host broadcasted a response before you had the chance to, unschedule your response

Recontextualizing L2/L3

- One IP, normally linked to one host, can be transformed at L2 into all hosts at a given subnet
  - This transformation is undetectable outside the subnet
- Other Uses
  - “All hosts” could also include “Many hosts” using L2 Multicast packets
  - Do we have another other situation where one IP “stands in” for many hosts?

NAT: Splitting IPs For Fun and Profit

- NAT multiplexes several hosts into one IP address by splitting on local port
  - Already munging IP, might as well munge ports too
  - Some implementations make best efforts to match local port inside the network w/ local port outside
    - Birthday Paradox: Collision chance = 1 / sqrt(range_of_local_ports) = 1/256
- If we can always match IP and Port, then we can always maintain end-to-end correctness
  - Only have a problem 1/256 connections to the same host
    - Alternate strategies exist – munge the SEQ#(problems w/ Window overlap), MTU decrement, **TIMESTAMPS**

MAC Address Translation

- "NATless NAT"
  - "Is it **possible** to share a globally addressable IP address without translating private IP ranges a la NAT?"
  - Is it **possible** to allow incoming connections to an IP multiplexed in this manner?
- Answer: Yes. Oh yes.
  - **NAT: L4->L3**
  - **ARP: L3->L2**
  - **MAT: L4->(L3,L2)**
    - **Multiplex with L2/L3 instead of just L3**
    - Make ARP Table dynamic, based on each individual L4 connection
    - Maintains L3 end-to-end integrity

Implementation: AllNewt 1.0

- "All New Translation Engine"
  - Another part of Paketto Keiretsu
- Translates arbitrary local IP addresses into globally routable IP addresses
  - Instead of just storing IP_SRC, stores IP_SRC, ETHER_DHOST, and ETHER_SHOST
  - If IP_SRC == External IP, packets will retain end-to-end integrity
  - If IP_SRC == RFC1918 IP, packets will be NATted normally
  - If IP_SRC == Yahoo/Microsoft/Whatever, packets will be NATted a little less normally
  - Multiple hosts can share the same IP address, if MAC is different(and vice versa -- Proxy ARP)

Pizza Protocol A La Mode

- “Anyone order a pizza?”
- Stateless approach: Ask everybody, drop RST|ACK, forward everything else.
  - Just broadcast to the IP
  - Actually works behind NATs, but you need to catalog all the local IPs
  - Drop all RSTs, pass all streams/ACKs
  - Breaks down when two people are listening on the same port
    - Can split port range(1022, 2022, 3022, etc. all being different instances of 22/ssh)
    - Apply host-level heuristics – priority for incoming selection based on outgoing sessions

Incoming State

- Stateful Approach ("you ordered the last one")
  - Ask everyone, but remember who's hosting
  - Send to the first host that replies
  - Increment the timer every time a packet is emitted from the serving host for that port
  - If no packets are emitted after a certain amount of time, allow open registration once more
- "It's amazing what you can do once security is not an issue."

TCP Splicing

- NAT Deadlock Resolution
  - "Is it **possible** to establish a TCP connection between two hosts, both behind NATs?"
- Answer: Yes…but it ain't pretty.
  - Convince each firewall that the other accepted the connection
    - Layers will need to be played against eachother to prevent certain otherwise desirable messaging behaviors from going too far

An Analogy

- Bill Gates ‘n Larry Ellison
  - Why? They can call anyone they want – their secretaries won’t stop ‘em.
  - None of us can call them – their secretaries will stop us.
  - If Bill or Larry did call us, they’d actually be able to hear us reply.
  - Asymmetry is in the initiation

Setting Up

- Alice and Bob both behind NATting firewalls
  - Firewalls authorize all outgoing sessions, block all incoming sessions
    - Block w/ state – no faking
    - Only accept fully validated responses to outgoing messages
      - Ports must match
      - SEQ#'s must match
    - Total outgoing trust, zero incoming trust

The Attempt

- Alice tries to send a message to Bob
  - SYN hits Alice's firewall, is given global IP + entry in state table "connection attempted"
  - SYN travels across Internet
  - SYN hits Bob's firewall, RST|ACK sent
  - RST|ACK hits Alice's firewall, entry in state table torn down, RST|ACK readdressed to Alice
  - Alice gets nowhere
- Bob does the same thing

Analysis

- Good
  - Entry in firewall state table, awaiting a reply
- Bad
  - Negative reply, entry in state table destroyed
- Can we get the former without the latter?

Doomed TTLs

- Packet first hits local firewall, gets NAT entry, travels across Internet, hits remote firewall, gets shot down.
  - Good stuff closer to us, bad stuff farther away
- TTL: Time To Live – SET TO ~4
  - Maximum number of hops packet is allowed to travel along the network before being dropped
  - Used by IP to prevent routing loops
  - Used by us to prevent state table from closing the hole
  - Alice SYNs w/ Doomed TTL
  - Bob SYNs w/ Doomed TTL
- Both firewalls have a hole open for eachother

Packets, Ports, Problems

- Three way handshake – SYN, SYN|ACK, ACK
  - Outgoing connections have SYNs and ACKs but no SYN|ACKs
- Ports
  - Need to agree on which ports are linking up
  - Need to discover firewall multiplexing rules
- Timing
  - Need to know when to attempt connection
- Solution to all three: Handshake Only Connection Broker
  - Involved only in setting up connection

The Other Shoe Drops

- Now you add a connection broker
  - HANDSHAKE ONLY.
- Sends the SYN|ACK Host/Port/SEQ# combination “virtually added” to firewall packet acceptance rules
  - Larry Ellison: “Bill Gates is going to call here in the next two minutes, please put his call through.”
- Need to generate packets, though

Local Port Strategies

- Some firewalls do best effort to match
- Some increment from a fixed counter
- Some use random local ports
  - Entropy cannot be differentiated – rule from kleptography
  - As long as it's translated back…
- Need to discover what strategy is being used

Full Broker Discovery

- Alice and Bob SYN Charlie 2x
- Charlie NFO Alice and Bob
- Alice and Bob SYN Charlie
- Alice and Bob DoomSYN Bob and Alice
- Alice and Bob SYN Charlie
- Charlie SYN|ACK Alice and Bob
  - Throw details about port selection in IPID
- Alice and Bob DoomACK Bob and Alice
- Alice and Bob begin normal TCP session to eachother, as if the other acknowledged correctly

Much easier strategies

- Source route through connection broker, drop the route once the connection goes live
- UDP NAT2NAT
  - Works all over the place in games
  - UDP is symmetrical – just spew packets at eachother with opposite local port / source port, and eventually the state system will assume the other's outgoing packet is a response to its own outgoing packet
  - You can run TCP over UDP
- Far less fun though

TTL-Based Firewall Analysis

- Emit a SYN with a low TTL
- SYN spawns ICMP error, hits local firewall, which rewrites IP header and forwards to local host
- Firewall **doesn't** rewrite ICMP data
  - Original outgoing header
  - **Can discover how firewall is munging our datastream**

State of Disarray

- State Management
- State = Buffers
  - Buffers need to be searched
  - Buffers need to be allocated
  - Buffers need to be overflown
    - If your name is Gobbles
- NAT normally needs to be stateful
  - A packet comes in, and given the Source IP, the Source Port, and the Destination Port, we check our tables to rewrite on the internal interface the Destination IP(not firewall) and maybe the destination port too
    - The MAC address is always rewritten, but with MAT we extract the correct MAC from the state table

Stateless NAT: Possible?

- State is all about things we have to remember
  - Stateless scanning is about extracting what we need from what we get back
- "Can we embed the NAT state in every outgoing IP packet such that every response received will contain the full NAT state"?
  - Answer: Yup. (Thanks, Spence.)
  - IP Timestamps Mode 3
    - IP Option against each host along the route. Up to four 4 byte IP addresses are specified, with space for up to four 4 byte timestamps to be added
    - If IP in the timestamp request matches IP of the router, the router replaces the timestamp with its own
    - If IP doesn't match, pass along the timestamps of others

Abusing IP Timestamps

- Insert timestamps from invalid IP's containing not actual timestamps but NAT state
- Encrypt NAT state so it may not be modified en route
- Decrypt NAT state upon packet return
- Problems
  - Need to insert IP options – may overflow packet, may need to fragment, etc.
  - IP options are sometimes blocked by firewalls
    - Damn source routers ;-)
- Possibilities with TCP Timestamps too
  - Reply field contains 32 bits of user specified stamp

Tricking Firewalls/IDSs

- Alice can forge a connection from an arbitrary IP by cooperating with Charlie
  - Alice looks like she's connecting to Yahoo, but is informing Charlie of the specifics of the connection attempt
  - Charlie replies as if he was Yahoo, and begins a TCP stream of arbitrary data to Alice from "Yahoo"
  - Alice acknowledges all data to "Yahoo" with the doomed TTL – we continue low TTL count through the data stream
- Really messy in terms of ICMP time exceeded messages, BUT logging systems might drop these messages

Interesting things are possible

- All code to be released at http://www.doxpara.com