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 2001
  - 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.

ObThreeWayHandshakeIntro

- 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.

What Do You Want?

- Port Ranges
  - 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

How Do You Want It?

- 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

SYN Cookies

- Developed in '96, when SYN floods became common
  - ACK reflects SEQ# of SYN|ACK
  - Encrypts connection state into the SYN|ACK's SEQ#
  - Therefore, you can use legitimate remote hosts – instead of kernel memory – to store handshake state
- Ahhh…but SYN|ACK also reflects SEQ# of SYN…

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
    - Verify SYN Cookie – did we actually scan this host?
    - Mark that port was up(SYN|ACK)or down(RST|ACK)

Observed Results

- Since no state is maintained within the scanner, we can send SYNs at wire speed
- 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 cookies made SYN reception more efficient
  - SYN|ACK cookies make SYN transmission more efficient

On Packet Structure

- Packets are “strangely ordered”
  - Next hop, previous hop, next protocol, next protocol, checksum, first hop, last hop, first app, last app, checksum, god knows what, checksum
- Why not sort everything? Why so much redundancy? Isn’t it inefficient?

Layers: Not What, But Who

- One medium, many messages
  - Listeners reconstruct meanings relevant to themselves, ignore the rest
  - Managed 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

Layer Duties

- Layer 1: Medium
- Layer 2: Previous Hop <-> Next Hop
- Layer 3: First Hop <-> Last Hop
- Layer 4: Previous App <-> Next App
- Layer 5: First App <-> Last App

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

The Summoning

- 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
- 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
  - 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 true L2 Multicast packets
  - Do we have another other situation where one IP “stands in” for many hosts?

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.
  - Keep the external IP on any and all hosts behind the gateway
  - Use NAT-style state management
  - **Multiplex on Layer 2**
    - Make ARP Table dynamic, based on each individual connection
    - Maintains L3 end-to-end integrity

Managing Local Ports

- 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), use TCP Timestamps

The "Anyone Order A Pizza" Protocol

- Stateless approach: Ask everybody, drop RSTs, forward everything else.
  - Just broadcast to the IP
  - Actually works behind NATs, but you need to catalog all the local Ips
  - Breaks down badly when two people are listening on the same port
    - Can split port range(1022, 2022, 3022, etc. all being different instances of 22/ssh)

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, using a connection broker to coordinate port selection and tunnel/spoof SYN|ACKs
    - 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, elicits the rejection.
  - Good at the beginning of life, bad at end of life
  - So shorten the packet's lifespan and it never goes bad.
- TTL: Time To Live
  - 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

New Paradigm

- Now able to add Host/Port/SEQ# combinations 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

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

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

Sequence

- 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

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 using standard TCP splicing
- Alice continues to doom her acknowledgments to Yahoo, and Charlie keeps sending packets as Yahoo.

Conclusion

- Interesting things are possible
- All code available for download at http://www.doxpara.com