

Detecting Forged TCP Reset Packets

Nicholas Weaver

Robin Sommer

Vern Paxson

Acknowledgements

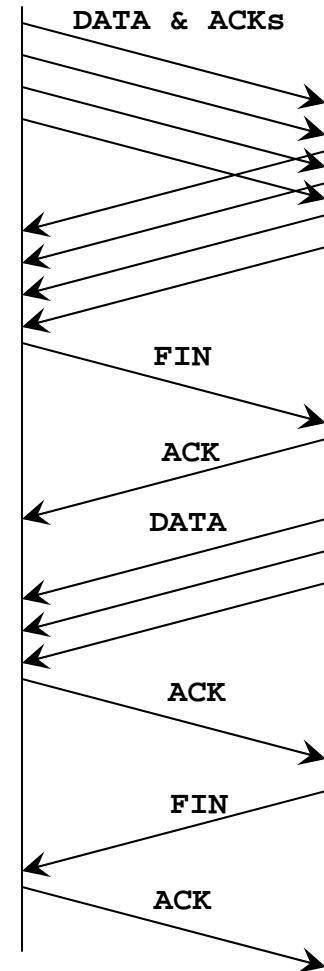
- Special thanks to those who ran our detector at their institutions:
 - Angelos Keromytis and Gabriela Cretu at Columbia
 - Angelos Stavrou at George Mason University
- Feedback from
 - Jim Mellander and Christian Kreibich
- Support from the National Science Foundation
 - CNS-0722035 and ITR/ANI-0205519
 - All opinions are those of the author, not the funder

Outline:

- Injected TCP Reset (RST) packets can be used for many purposes.
 - Our goal is *not* to judge their use but to make their use *transparent*
- TCP 101: TCP Reset (RST) Packets
 - Network Management 101: Injected RSTs
- Injected Packets: Constraints and Freedoms
- Detecting Injected Packets: Race Conditions
- Fingerprinting Packet Injectors
- What sources did we see?
 - TCP Packet Injectors and their uses
 - Non-injected sources

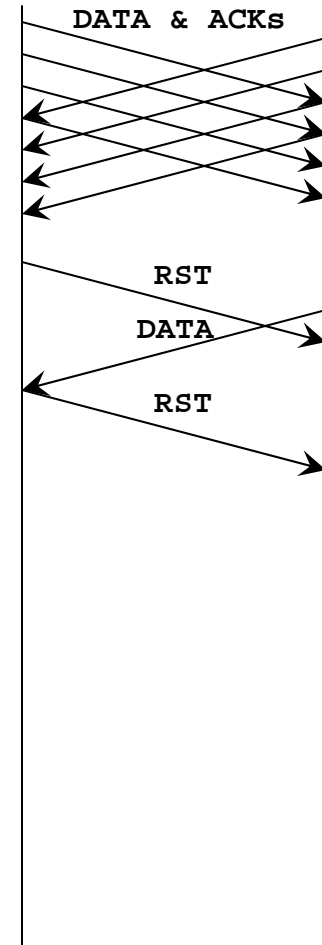
TCP 101: Connection Termination

- A side sends a TCP Finish (FIN) to indicate that it is done sending but not receiving
 - Resulting connection is “half-closed”
- Connection is only closed when the other side sends a FIN of its own
 - Until then, the other side can keep sending data



TCP 101: Connection Aborting

- But what if a side does not want to send *or receive* any more data:
 - Program closed
 - Abort the connection
 - Deny the connection after first accepting it
- A TCP Reset (RST) tells the other side of the connection:
 - There will be no more data from this source on this connection
 - This source will not accept any more data, so no more data should be sent
- Once a side has decided to abort the connection, the *only* subsequent packets sent on this connection may be RSTs in response to data
 - Once a side accepts a RST, it will no longer send data or accept any more data
- Yet RSTs are quite common, 10-15% of ALL flows are terminated by a RST rather than a FIN
 - For HTTP, it can be over 20%



Network Management 101: Connection Blocking

- Many reasons to terminate a connection:
 - Required network censorship (the “Great Firewall” of China)
 - Blocking “undesirable” protocols (blocking P2P traffic)
 - Stopping spam and network attacks
- Can build either an in-path device or an out-of-path device
 - In path devices can just drop traffic:
But they are dangerous! They add points-of-failure and can slow down the network
- An out-of-path device is simpler to build, but you have to terminate the flow somehow:
 - Tell an in-path device to block a flow (ACL injection)
 - Send bogus TCP data or FINs
 - May result in packet storms
 - Send bogus TCP RSTs
 - If **one** side accepts the packet, the connection will terminate
 - Injecting RSTs is the generally most preferred method by network engineers

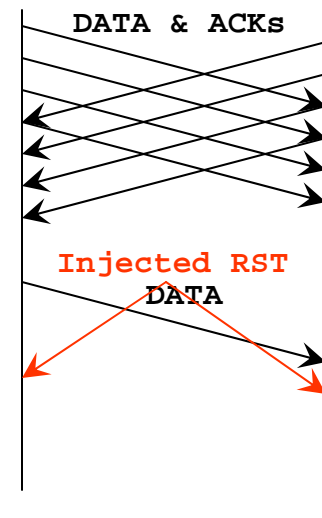
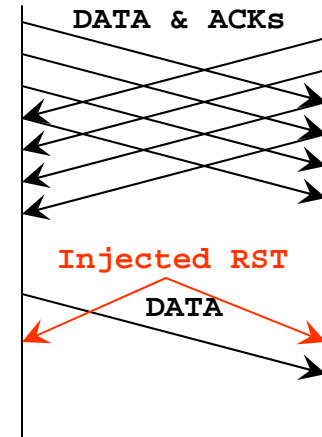
QuickTime™ and a decompressor are needed to see this picture.

What Can an Injected TCP RST Look Like?

- The 5-tuple (source and destination ports and IP addresses) must be correct
 - Can send to both directions, to ensure that one side accepts the RST
- The packet must have **consistent** sequencing:
 - Many TCP stacks will accept any RST **in window**
 - Paranoid stacks will only accept RSTs **in sequence**
 - Prevents blind TCP RST injection
- Almost complete freedom elsewhere
 - TTL may be different (because the injected packet took a different path)
 - But TTL can be highly variable on normal RSTs too
 - The ACK field is not checked
 - IPID, other TCP flags (ACK flag, ECN, etc)
- Yet we'd expect an end host or injector to be **consistent** in how it creates packets

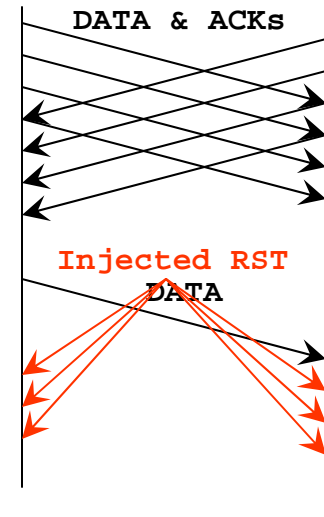
RST Injection Race Conditions

- An injected reset is an **additional** packet, it can't remove a packet from the network
- Unavoidable race conditions which create detectable out-of-spec packet flows:
- DATA_SEQ_RST:
 - A data packet immediately following a RST packet where data packet (seq + len) > RST seq
 - Caused by a subsequent data packet in flight
- RST_SEQ_DATA:
 - A RST packet immediately following a data packet where RST seq < data packet (seq + len)
 - Caused because the injector was too slow in sending the packets



Race Conditions Cause RSTs to be ignored

- RST_SEQ_DATA creates RSTs that are ignored
 - So countermeasure is to send multiple RSTs with increasing sequence number
 - Thus the second or third RST should be in-window
 - Best increment: size of last packet
 - Second-best increment: standard MTU
- RST_SEQ_CHANGE:
 - Back to back RST packets where the second RST seq \neq first and RST seq $>$ maximum sent sequence
RST seq $>$ maximum received ACK
 - (In case we missed a packet or the other side is not following the specification)
- Non-robust injectors can only be detected when the race conditions occur
- Injectors robust the the RST_SEQ_DATA race condition can ***always*** be detected



RST Injectors Also Create “Interesting” Aborts

- SYN_RST:
 - A RST packet immediately following a SYN packet
 - Note that web browsers and SMTP authentication clients do this for benign reasons:
For example, the user misclicks on a bookmark and then immediately hits “STOP”
- SYN_ACK_RST:
 - A RST packet immediately following the SYN/ACK from the server
 - Note that web servers and SMTP servers do this for benign reasons:
For example, accept a connection and **then** check for presence on a blacklist
- RST_ACK_CHANGE:
 - Back to back RST packets where the second one’s ACK != the first one’s ACK and the ack doesn’t make sense (greater than any seen packet in the other direction, not equal to the SEQ, not equal to zero)
 - An identified injector does this

The High Level Procedure

- A ***click*** element to ***passively*** detect suspicious RST packets as they occur
 - Either as a live network monitor or on packet traces
 - Extract ***context*** around every suspicious packet for further analysis
- Postprocess captured packets to remove private information
 - Strip the payload from the packets
 - Perform hostname and GeoIP lookups
 - Optional anonymization
- Place alerts in a database and look for fingerprints and other commonalities between alerts
 - Fingerprints generated by manual examination of clusters of alerts

More Details

- The click module used a small (256k entry) flow cache
 - 32 way associative, evict-oldest policy
 - Time window is **not** a problem: injected RSTs must be close to the associated packet to be effective
 - RST_SEQ_DATA, DATA_SEQ_RST, RST_SEQ_CHANGE, and RST_ACK_CHANGE are set with a threshold of 2 seconds
 - SYN_RST and SYN_ACK_RST are set with a threshold of .1 seconds
- Buffer 256K packets and isolate any “interesting” host-pairs in the buffer
 - Allows a one-pass procedure to capture the context of an alert
- Associate reverse name lookup and GeoIP information with each IP
- Optional anonymization pass:
 - replace IP with random ID, remove hostname from FQDN
- Place all alerts and all packet headers -200 to +100 around each alert into the database for analysis
- Ran on 4 networks in early 2008:
 - Operationally at ICSI for months, 19 hours at UC Berkeley, 24 hours at Columbia CS, 5 hours at George Mason University

The Comcast Sandvine Injector

- A multiple-packet RST injector with a distinct fingerprint:
 - First RST packet: $\text{ipid} += 4$
 - Second RST packet: $\text{ipid} += 1$
 $\text{sequence} += 12503$
 - Large increment is a known bug to Sandvine, it should be smaller
- Numerous alerting IP addresses
 - 106 communicating with ICSI, 30 communicating with Berkeley, 36 communicating with Columbia, 2 communicating with GMU
 - Most of the ICSI alerts correspond to known incidents of unauthorized P2P usage
- Comcast is **not** the only user of this tool
 - Cox: 35 at ICSI, 262 at Berkeley, 3 at Columbia
 - Unknown Korean ISP: 1 at ICSI, 50 at Berkeley, 4 at Columbia
 - 2 other alerts with no reverse name lookup

Is Comcast Only Blocking Leeches?

- Comcast made public statements that they were only blocking uploads from Comcast peers (“Seeding” and “leeches”)
 - Blocking leeches and incidental seeding directly benefits Comcast’s customer (although hurts BitTorrent overall)
 - Blocking deliberate seeding penalizes Comcast’s customer
 - Problem of transparency: if you **know** the policy is “no seeding” there are easy workarounds for legal content
- Looked at flows at ICSI where we see the SYN and blocking RST
 - All but 7% are clear seeds/leeches
 - For remaining 7%, Sandvine supports recognizing pure seeds by looking at the initial BitTorrent message

QuickTime™ and a decompressor are needed to see this picture.

The Bezeq International and IPID 256 Injectors on P2P traffic

- Bezeq International (Israeli Telecom/Cable company) disrupting P2P traffic
 - Common at ICSI (25 alerting IPs), seen at Columbia (2 alerting IPs)
 - Multiple RST packets with a distinct fingerprint:
 - Always IPID = 16448 (0x4040)
 - Second and successor packets increment **ack** field, not the **seq** field
 - Assume to be a bug
- Korean IPID 256 injector
 - Single packet injector, IPID = 256
 - Single packet injectors are less robust but somewhat less detectible
 - 9 alerting IPs seen at ICSI, 90 alerting IPs to Berkeley, 16 alerting IPs to Columbia
 - Plus 5 alerts at Berkeley to other Asian countries

Gummadi et al

Report on BitTorrent Blocking

- Gummadi et al built and used a Java test-client
 - Java client emulates a BitTorrent transfer, checks for some seeding/blocking policies
 - Requires transferring almost 30 MB of data for the full test
- They discovered three ISPs performing significant blocking: Comcast, Cox, and StarHub (Singapore)
- We can confirm StarHub (maxonline.com.sg) was blocking P2P traffic
 - We see 4 alerting IPs from this ISP at ICSI which appear to be a multipacket injector:
 - Second RST's sequence increment is equal to the last data packet's length
 - 34 flows show interference
 - But we were unable to develop a better fingerprint for this injector

Spam and Virus

Blocking with RST injection

- yournet.ne.jp: Apparently blocking Spam Bots
 - 29 IPs generating SYN_RST alerts on port 25 to ICSI
 - >30% of **all** IPs generating SYN_RST alerts for SMTP to ICSI
 - TTL is usually +5, but not always. IPID appears unrelated
 - Appears to be a dynamic spam-blocking system
 - Rather than just block outbound port 25:
Heuristically detect spam bots and then block their messages with RST packets
- Uvic.ca: Apparently blocking viruses
 - One smtp server attempting to forward a MyDoom bounce message back to ICSI:
Message is blocked with a series of RST packets
 - ~10 RST packets, increment sequence by 1500, IPID = 305, TTL 38 higher
 - Mail server then retries a few hours later, and the same thing occurs
 - Timed out after several days

The Great Firewall of China

- Appears to be multiple injectors with distinct fingerprints:
 - IPID 64: Multiple packets, IPID always 64
 - IPID -26: IPID is 26 less than previous packet
 - SEQ 1460: Multiple packets, always increments by 1460, unrelated IPID
 - RAE: Single packet, sets both ACK bit and ECN Nonce bit!?!?
- Multiple injectors can be seen on the same flow!
 - 102 hosts at ICSI show multiple chinese fingerprints: redundant devices along the path?!
 - Although the RAE injector appears to be distinct, only 2 overlaps at ICSI
 - One web request from columbia shows:
 - IPID 64 injector RSTs, then (probably) the 1460 injector, then a RST from the host, then a series of RSTs from the IPID -26 injector **whose IPID seems derived from the 1460 injector's RST packet!?!?**
 - Or perhaps our fingerprints are too specific: a single injector could have multiple fingerprints?

But Not All “Suspicious” RSTs are Injected!

- NATs can generate spurious RSTs
 - And bad ones too, in active flows...
- Google and Yahoo’ load balancers occasionally generate RST_SEQ_DATA and DATA_SEQ_RST alerts
- Planetlab is awful: generates RST_SEQ_DATA and DATA_SEQ_RST errors **all the time**
 - We excluded Planetlab from our datasets, after a 1 hour trace at Columbia generated 300 alerts on Planetlab communication!
- Random out-of-sequence RSTs with IPID=0 in the middle of traffic
 - Including internal hosts. Bad NATs? Bad Endhosts?
- Common SYN_RST behavior with no geographic commonality
 - TTL > 128 higher or IPID = 65259
 - Bad NATs? Bad Endhosts?
- Thus until the alerts are correlated in a database and fingerprinted, just alerting is insufficient to conclude interference

Conclusions

- Can ***detect*** injected TCP RST packets
 - The same technique can be used for other packet-injection attacks: we have such an IDS detector for DNS attacks
- Can ***fingerprint*** many sources of injected TCP RST packets
- Many ***benign*** sources of seemingly injected RST packets
 - Without fingerprints or correlation, can't conclude that suspicious RSTs are actually injected by a network management process
- Email nweaver@icsi.berkeley.edu if you desire a copy of the source code

Backup: All Fingerprints

QuickTime™ and a
decompressor
are needed to see this picture.

Backup: Identified Sources

QuickTime™ and a
decompressor
are needed to see this picture.

Backup: Identified Benign Sources

QuickTime™ and a
decompressor
are needed to see this picture.