



DDoS protection

Using Netfilter/iptables

Jesper Dangaard Brouer
Senior Kernel Engineer, Red Hat

KLID 31st March 2014

Email: brouer@redhat.com / netoptimizer@brouer.com / hawk@kernel.org

Who am I

- Name: Jesper Dangaard Brouer
 - Linux Kernel Developer at Red Hat
 - Edu: Computer Science for Uni. Copenhagen
 - Focus on Network, Dist. sys and OS
 - Linux user since 1996, professional since 1998
 - Sysadm, Kernel Developer, Embedded
 - OpenSource projects, author of
 - ADSL-optimizer, CPAN IPTables::libiptc, IPTV-Analyzer
 - Patches accepted into
 - Linux kernel, iproute2, iptables, libpcap and Wireshark
 - Organizer of Netfilter Workshop 2013



What will you learn?

- Linux Kernel is vulnerable to simple SYN attacks
- End-host mitigation's already implemented in kernel
 - show it is not enough
- Kernel: serious "listen" socket scalability problem
 - solution is stalled ... how to work-around this
- Firewall-based solution: synproxy (iptables/netfilter)
- How fast is stateful firewalling
 - Where is our pain points
 - Learn Netfilter tricks: boost performance a factor 20



First: Basic NIC tuning 101

- All tests in presentation
- Basic tuning
 - First kill “irqbalance”
 - NIC hardware queue, are CPU aligned
 - Disable Ethernet flow-control
 - Intel ixgbe hw/driver issue
 - single blocked hw queue blocks others
 - Fix in kernel v3.5.0 commit 3ebe8fdeb0 (ixgbe: Set Drop_EN bit when multiple Rx queues are present w/o flow control)



Focus: Flooding DoS attack

- **Denial of Service (DoS) attacks**
- **Focus: TCP flooding attacks**
 - Attacking the 3-**Way HandShake** (3WHS)
 - End-host resource attack
 - SYN flood
 - SYN-ACK floods
 - ACK floods (3rd packet in 3WHS)
 - Attacker often spoofs src IP
- Described in [RFC 4987](#):
TCP SYN Flooding Attacks and Common Mitigations



Linux current end-host mitigations

- Jargon RFC 4987 (TCP SYN Flooding Attacks and Common Mitigations)
- Linux uses hybrid solution
 - SYN “cache”
 - Mini request socket
 - Minimize state, delay full state alloc
 - SYN “backlog” of outstanding request sockets
 - Above limit, use SYN “cookies”



Details: SYN "cache" savings

- Small initial TCB (Transmission Control Block)
- struct request_sock (size 56 bytes)
 - mini sock to represent a connection request
- But alloc size is 112 bytes
 - SLAB behind have sizeof(struct tcp_request_sock)
 - Structs embedded in each-other
 - 56 bytes == struct request_sock
 - 80 bytes == struct inet_request_sock
 - 112 bytes == struct tcp_request_sock
- Full TCB (struct inet_sock) is 832 bytes
(note, sizes will increase/change in more recent kernels)



Details: Increasing SYN backlog

- Not recommended to increase for DoS
 - Only increase, if legitimate traffic cause log:
 - “TCP: Possible SYN flooding ...”
- Increasing SYN backlog is not obvious
 - Adjust all these:
 - /proc/sys/net/ipv4/tcp_max_syn_backlog
 - /proc/sys/net/core/somaxconn
 - Syscall listen(int sockfd, int **backlog**);



SYN cookies

- Simplified description
 - SYN packet
 - don't create any local state
 - SYN-ACK packet
 - Encode state in SEQ# (and TCP options)
 - ACK packet
 - Contains SEQ#+1 (and TCP timestamp)
 - Recover state
 - SHA hash is computed with local secret
 - Validate (3WHS) ACK packet state



Details: SYN-cookies

- SYN cookies SHA calculation is expensive
- SNMP counters (Since kernel v3.1)
 - **TCPReqQFullDoCookies** : number of times a SYNCOOKIE was replied to client
 - **TCPReqQFullDrop** : number of times a SYN request was dropped because syncookies were not enabled.
- Always on option
 - `/proc/sys/net/ipv4/tcp_syncookies = 2`



So, what is the problem?

- Good End-Host counter-measurements
- Problem: LISTEN state scalability problem
 - Vulnerable for all floods
 - SYN, SYN-ACK and ACK floods
- Numbers: Xeon CPU X5550 10G ixgbe
 - NO LISTEN socket:
 - 2.904.128 pkts/sec -- SYN attack
 - LISTEN socket:
 - 252.032 pkts/sec -- SYN attack
 - 336.576 pkts/sec -- SYN+ACK attack
 - 331.072 pkts/sec -- ACK attack



Problem: SYN-cookie vs LISTEN lock

- Main problem:
 - SYN cookies live under LISTEN lock
- I proposed SYN brownies fix (May 2012)
 - <http://thread.gmane.org/gmane.linux.network/232238>
 - Got rejected, because not general solution
 - e.g. don't handle SYN-ACK and 3WHS
 - NFWS2013 got clearance as a first step solution
 - Need to “forward-port” patches
 - (Bug 1057364 - RFE: Parallel SYN cookies handling)

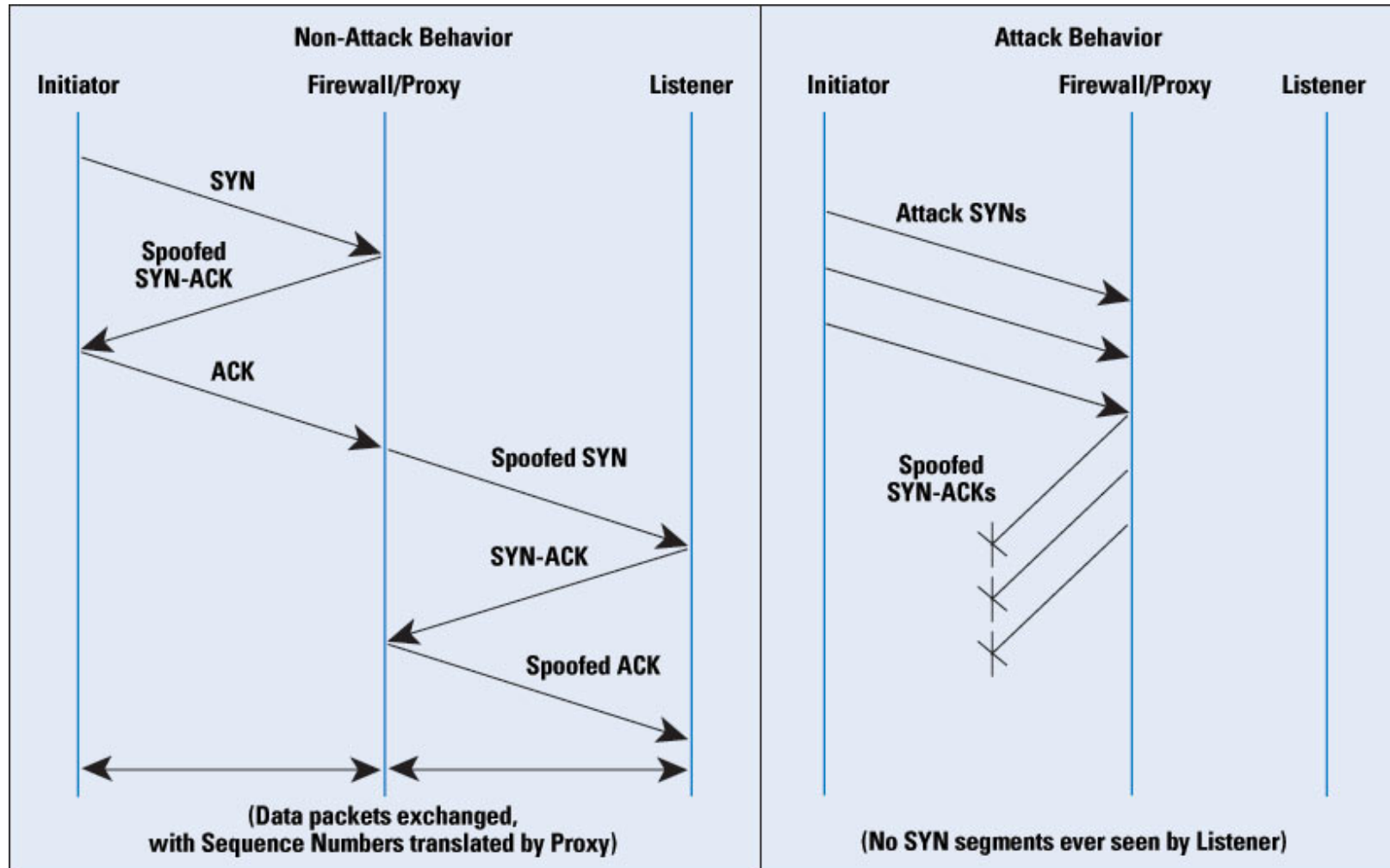


Firewall and Proxy solutions

- **Network-Based Countermeasures**
 - Wesley M. Eddy, describes SYN-proxy
 - In Cisco: The Internet Protocol Journal - Volume 9, Number 4, 2006, link: <http://goo.gl/AC1AAZ>
 - Netfilter: iptables target **SYNPROXY**
 - Avail in kernel 3.13 and RHEL7
 - By Patrick McHardy, Martin Topholm and Me
 - Also works on localhost
 - General solution
 - Solves SYN and ACK floods
 - Indirect trick also solves SYN+ACK



SYN proxy concept



Conntrack performance(1)

- SYNPROXY needs conntrack
 - Will that be a performance issue?
- Base performance:
 - 2.904.128 pkts/sec -- NO LISTEN sock + no iptables rules
 - 252.032 pkts/sec -- LISTEN sock + no iptables rules
- Loading conntrack: (SYN flood, causing new conntrack)
 - 435.520 pkts/sec -- NO LISTEN sock + **conntrack**
 - 172.992 pkts/sec -- LISTEN sock + **conntrack**
- Looks bad...
 - but I have some tricks for you ;-)



Conntrack performance(2)

- Conntrack (lock-less) **lookups are *really* fast**
 - Problem is insert and delete conntracks
 - Use to protect against SYN+ACK and ACK attacks
- Default netfilter is in TCP “loose” mode
 - Allow ACK pkts to create new connection
 - Disable via cmd:

```
sysctl -w net/netfilter/nf_conntrack_tcp_loose=0
```
- Take advantage of state “INVALID”
 - Drop invalid pkts *before* reaching LISTEN socket
 - ```
iptables -m state --state INVALID -j DROP
```





# Conntrack perf(3) ACK-attacks

- **ACK attacks**, conntrack performance
- Default “loose=1” and pass INVALID pkts
  - 179.027 pkts/sec
- Loose=0 and and pass INVALID pkts
  - 235.904 pkts/sec (listen lock scaling)
- Loose=0 and and DROP INVALID pkts
  - 5.533.056 pkts/sec



# Conntrack perf(4) SYN-ACK attack

- **SYN-ACK attacks**, conntrack performance
  - SYN-ACKs don't auto create connections
  - Thus, changing “loose” setting is not important
- Default pass INVALID pkts (and “loose=1”)
  - 230.348 pkts/sec
- Default DROP INVALID pkts (and “loose=1”)
  - 5.382.265 pkts/sec
- Default DROP INVALID pkts (and “loose=0”)
  - 5.408.307 pkts/sec



# Synproxy performance

- **Only conntrack SYN attack problem left**
  - Due to conntrack insert/delete lock scaling
    - (will be fixed in next kernel v3.15)
- **Base performance:**
  - 244.129 pkts/sec -- LISTEN sock + no iptables rules
- **Loading conntrack: (SYN flood, causing new conntrack)**
  - 172.992 pkts/sec -- LISTEN sock + **conntrack**
- **Using SYNPROXY**
  - **2.869.824** pkts/sec -- LISTEN sock + **synproxy** + conntrack
    - Parallel SYN cookies
    - Delay creating conntrack until 3WHS-ACK



# iptables: synproxy setup(1)

Using SYNPROXY target is complicated

- SYNPROXY works on untracked conntracks

In “raw” table, “notrack” SYN packets:

```
iptables -t raw -I PREROUTING -i $DEV -p tcp -m tcp --syn \
--dport $PORT -j CT --notrack
```



# iptables: synproxy setup(2)

- More strict conntrack handling
  - Need to get unknown ACKs (from 3WHS) to be marked as INVALID state
    - (else a conntrack is just created)

Done by sysctl setting:

```
/sbin/sysctl -w net/netfilter/nf_conntrack_tcp_loose=0
```



# iptables: synproxy setup(3)

- Catching state:
  - UNTRACKED == SYN packets
  - INVALID == ACK from 3WHS

## Using SYNPROXY target:

```
iptables -A INPUT -i $DEV -p tcp -m tcp --dport $PORT \
-m state --state INVALID,UNTRACKED \
-j SYNPROXY --sack-perm --timestamp --wscale 7 --mss 1460
```



# iptables: synproxy setup(4)

- Trick to catch SYN-ACK floods
  - Drop rest of state INVALID, contains SYN-ACK

```
iptables -A INPUT -i $DEV -p tcp -m tcp --dport $PORT \
-m state --state INVALID -j DROP
```

- Enable TCP timestamping
  - Because SYN cookies uses TCP options field

```
/sbin/sysctl -w net/ipv4/tcp_timestamps=1
```



# iptables: synproxy setup(5)

- Conntrack entries tuning

- Max possible entries 2 Mill

- $288 \text{ bytes} * 2 \text{ Mill} = 576.0 \text{ MB}$

- ```
net/netfilter/nf_conntrack_max=2000000
```

- **IMPORTANT:** Also adjust hash bucket size

- `/proc/sys/net/netfilter/nf_conntrack_buckets` writeable

- via `/sys/module/nf_conntrack/parameters/hashsize`

- Hash $8 \text{ bytes} * 2\text{Mill} = 16 \text{ MB}$

- ```
echo 2000000 > /sys/module/nf_conntrack/parameters/hashsize
```





# Performance SYNPROXY

- Script `iptables_synproxy.sh` avail here:
  - [https://github.com/netoptimizer/network-testing/blob/master/iptables/iptables\\_synproxy.sh](https://github.com/netoptimizer/network-testing/blob/master/iptables/iptables_synproxy.sh)
- Using SYNPROXY under attack types:
  - 2.869.824 pkts/sec – SYN-flood
  - 4.948.480 pkts/sec – ACK-flood
  - 5.653.120 pkts/sec – SYN+ACK-flood

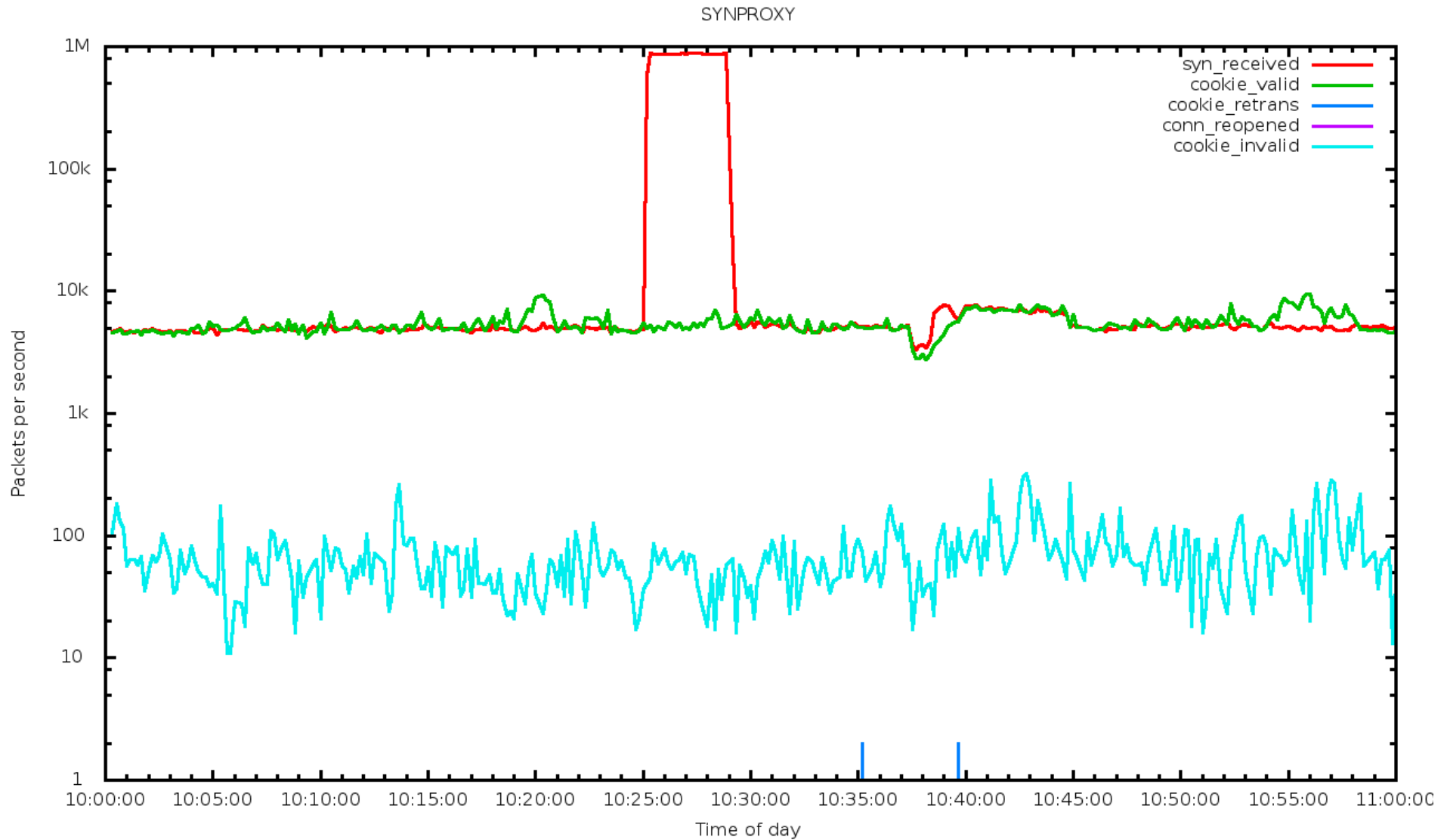


# SYNPROXY parameters

- The parameters given to SYNPROXY target
  - Must match the backend-server TCP options
  - Manual setup (helper tool nfsynproxy)
  - Only one setting per rule
  - Not useful for DHCP based network
- *Future plan*
  - Auto detect server TCP options
  - Simply allow first SYN through
    - Catch SYN-ACK and decode options
    - (RHBZ 1059679 - RFE: Synproxy: auto detect TCP options)

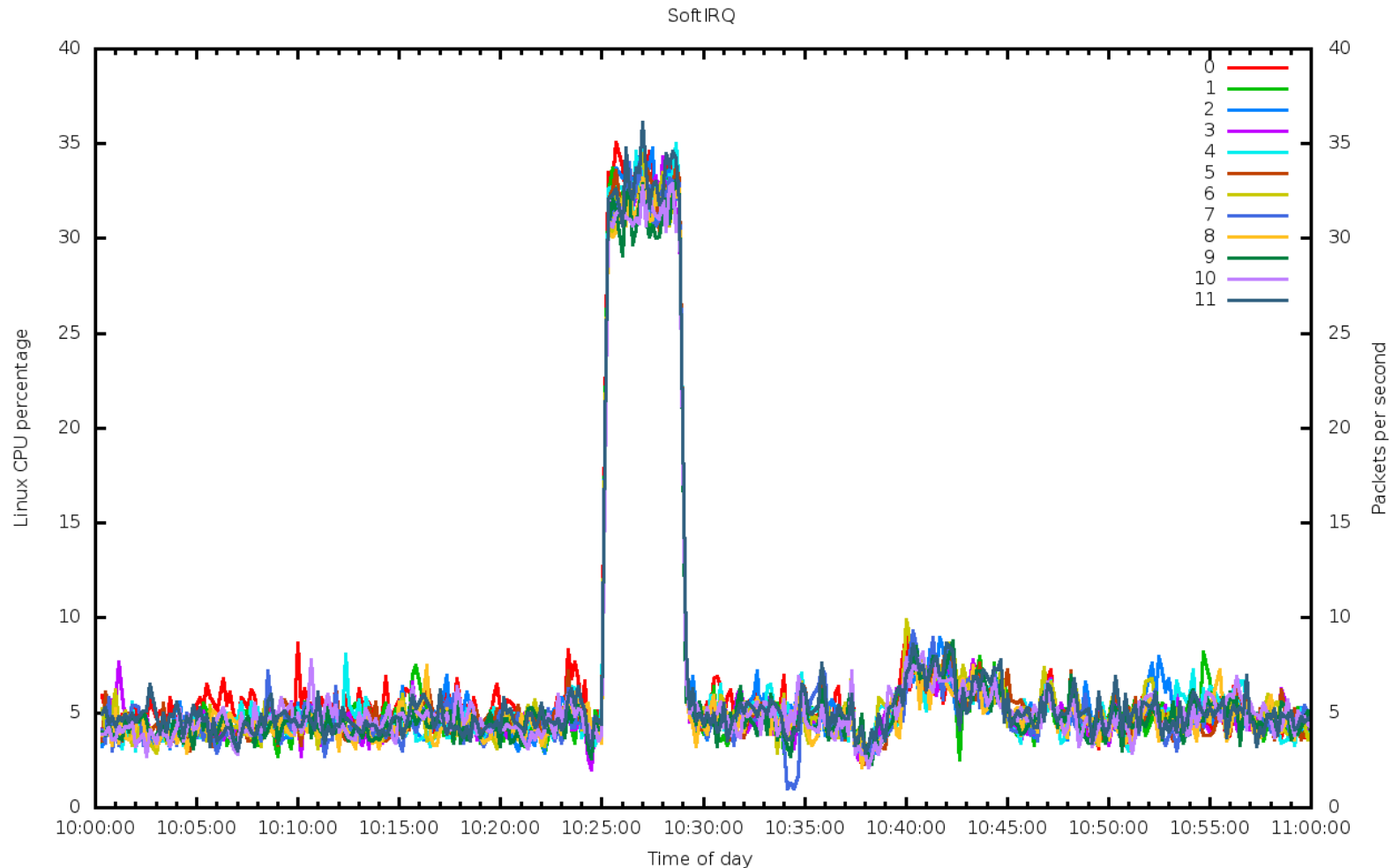


# Real-life(1): Handle 900 Kpps

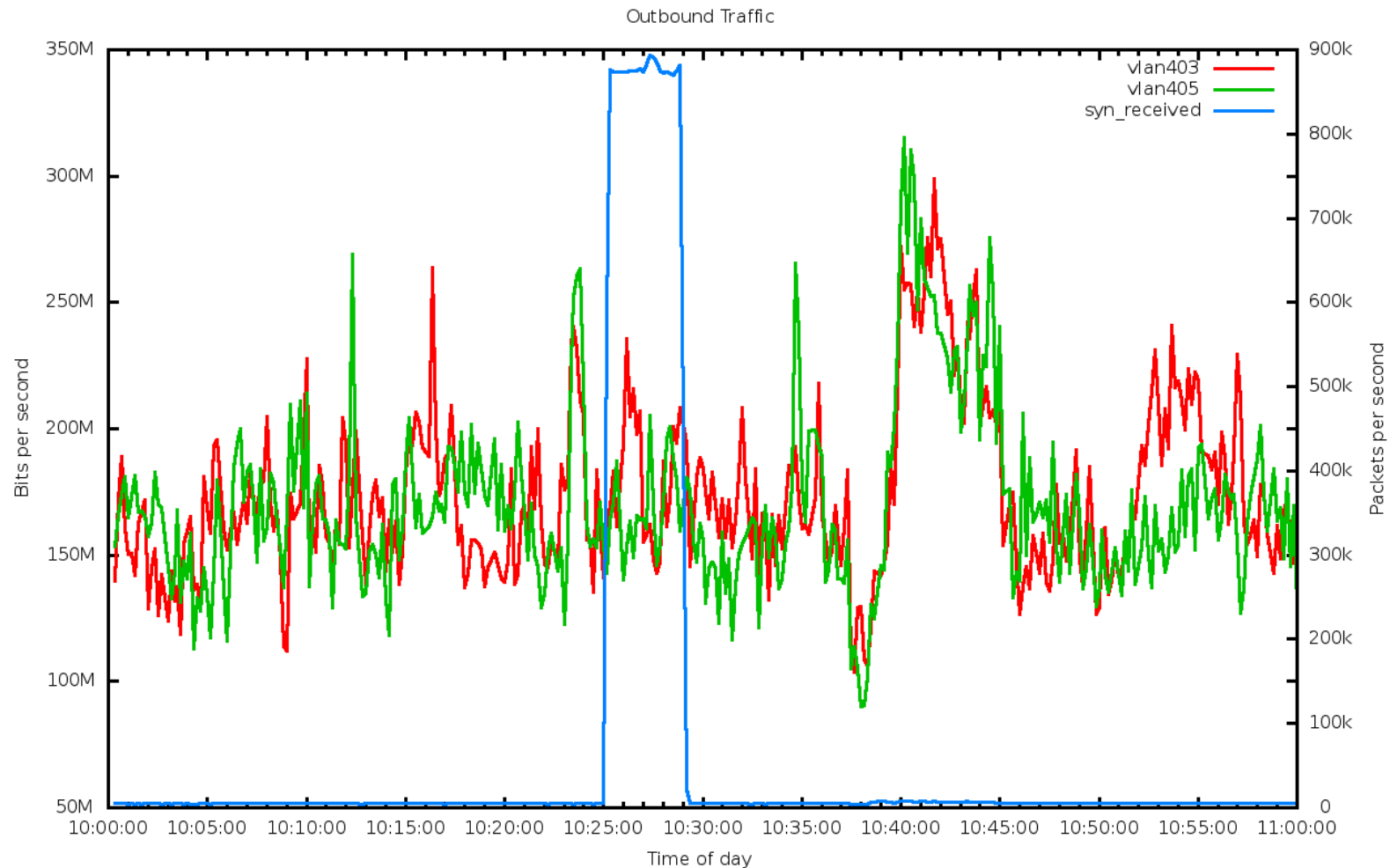


# Real-life(2): SHA sum expensive

- SYN cookie SHA sum is expensive
  - Bug 1057352 - RFE: Improve SYN cookies calculations



# Real-life(3): Out traffic normal



# Issue: Full connection scalability

- Still exists: Scalability issue with full conn
  - Made it significantly more expensive for attackers
    - (they need real hosts)
- Future work: fix scalability for
  - Central lock: LISTEN socket lock
  - Central lock: Netfilter new conntracks



# Fixing central conntrack lock

- Conntrack issue
  - Insert / delete conntracks takes central lock
  - Removed this central lock
    - My patches have been accepted
    - Fix on route to next kernel v3.15
    - ([RHBZ 1043012](#) - "netfilter: conntrack: remove the central spinlock")
- Results, SYN-flood
- No LISTEN socket to leave out that issue
  - 435.520 pkts/sec – conntrack with central lock
  - 1.626.786 pkts/sec – conntrack with parallel lock



# Hack: Multi listen sockets

- Hack to work-around LISTEN socket lock
  - Simply LISTEN on several ports
  - Use iptables to rewrite/DNAT to these ports

```
iptables -t nat -A PREROUTING -p tcp --dport 80 \
-m cpu --cpu 0 -j REDIRECT --to-port 8080
```

```
iptables -t nat -A PREROUTING -p tcp --dport 80 \
-m cpu --cpu 1 -j REDIRECT --to-port 8081
```





# Hack: Full conn hashlimit trick(1)

- Problem: Full connections still have scalability
- Partition Internet in /24 subnets
  - $(128*256*256 / 2097152 = 4 \text{ max hash list})$
- Limit SYN packets e.g. 200 SYN pps per src subnet
- Mem usage: fairly high
  - Fixed: htable-size  $2097152 * 8 \text{ bytes} = 16.7 \text{ MB}$
  - Variable: entry size  $104 \text{ bytes} * 500000 = 52 \text{ MB}$
- Issue: Hashlimit needs scalability fix
  - (lock on new entries, e.g. subnet not seen before)



# Hack: Full conn hashlimit trick(2)

- Using hashlimit as work-around
  - Attacker needs many real hosts, to reach full conn scalability limit

```
iptables -t raw -A PREROUTING -i $DEV \
-p tcp -m tcp --dport 80 --syn \
-m hashlimit \
--hashlimit-above 200/sec --hashlimit-burst 1000 \
--hashlimit-mode srcip --hashlimit-name syn \
--hashlimit-htable-size 2097152 \
--hashlimit-srcmask 24 -j DROP
```



# Alternative usage of "socket" module

- Avoid using conntrack
  - Use xt\_socket module
    - For local socket matching
    - Can filter out 3WHS-ACKs (and other combos)
      - Parameter --nowildcard
      - Problem can still be invalid/flood ACKs
      - Mitigate by limiting e.g.hashlimit
    - Didn't scale as well as expected
- [https://github.com/netoptimizer/network-testing/blob/master/iptables/iptables\\_local\\_socket\\_hack.sh](https://github.com/netoptimizer/network-testing/blob/master/iptables/iptables_local_socket_hack.sh)



# The End

- Thanks to Martin Topholm and One.com
  - For providing real-life attack data
- Download slides here:
  - <http://people.netfilter.org/hawk/presentations/>
- If unlikely(time for questions)
  - Questions?



# Extra Slides



# Disable helper auto loading

- Default is to auto load conntrack helpers
  - It is a security risk!
    - Poking holes in your firewall!
  - Disable via cmd:

```
echo 0 > /proc/sys/net/netfilter/nf_conntrack_helper
```
- Controlled config example:

```
iptables -t raw -p tcp -p 2121 -j CT --helper ftp
```
- Read guide here:  
<https://home.regit.org/netfilter-en/secure-use-of-helpers/>

