Complexity Attacks on ipfw

David Malone <dwmalone@freebsd.org>
Josh Tobin <rjt@maths.tcd.ie>

20 September 2009
Complexity Attacks

- Working on some code — too slow.
- You implement some clever algorithm — nice and fast.
- Algorithm Performance: average and worst case.
- E.g. Quicksort.
- Who controls what performance you get?

Hash Table

Lookup scheme to avoid cost of searching full list.

carrot zuchinni... ...apricot apple

becomes:

\[ a \text{ apricot, apple} \]
\[ b \text{ banana} \]
\[ c \text{ carrot} \]
\[ \ldots \]

\[ z \text{ zuchinni} \]

Hash function \( h(x) \), XOR typical. Cost: \( O(N) \rightarrow O(N/H) \).
Algorithmic Attacks

Suppose attacker controls keys.

a abduce, abducens, abducent, abduct, abduction, abductor ...

b

c

z

Attacker finds $x_i$ so that $h(x_1) = h(x_2) = \ldots = h(x_i)$. 
Crosby and Wallach

**Bro** XOR based hash to store state. Dual core Pentium drop traffic using 30s of 160kbps or 6 minutes of 16kbps.

**Perl** Hash used for (uh) Hashes. Insert 90k short strings. Usually took under 2s. With crafted input took almost 2 hours.

Also attacks on directory cache, Python, GLIB, Java, ... Related attacks on browsers, web servers, ...
Fixes?

- Don’t use hashes, choose good worst case performance.
- — can be hard to fit into existing code.
- Use cryptographic hashes.
  - — output truncated
- Use keyed hash.
  - — sometimes use misnomer “universal”.
ipfw Flow Lookup

- Long standing firewall in FreeBSD.
- About 4.0 it grew stateful features.
- Lookup single flow by tuple (src IP, dst IP, src port, dst port).
- Uses hash table as optimisation.
- For IPv4 96 bits of input.
- For IPv6 288 bits of input.
- Note inexact flow matching different!
Demonstration attack
Xor
\[ h \leftarrow 0 \]
foreach (byte[i]) \[ h \leftarrow h \oplus \text{byte}[i] \]
return h

DJB2
\[ h \leftarrow 5381 \]
foreach (byte[i]) \[ h \leftarrow 33 \times h + \text{byte}[i] \]
return h

XorSum
\[ h \leftarrow 0 \]
foreach (byte[i]) \[ h \leftarrow h + (\text{byte}[i] \oplus K[i]) \]
return h

SumXor
\[ h \leftarrow 0 \]
foreach (byte[i]) \[ h \leftarrow h \oplus (\text{byte}[i] + K[i]) \]
return h

Universal
\[ h \leftarrow 0 \]
foreach (byte[i]) \[ h \leftarrow h + K[i] \times \text{byte}[i] \]
return h \mod 65537

Pearson
\[ h_1 \leftarrow h_2 \leftarrow 0 \]
foreach (byte[i]) \[ h_1 \leftarrow T_1[\text{byte}[i] \oplus h_1] \]
\[ h_2 \leftarrow T_2[\text{byte}[i] \oplus h_2] \]
return \[ h_1 + h_2 \times 256 \]

MD5
return two bytes of MD5(bytes)

SHA
return two bytes of SHA(bytes)
Other options

Don’t need to use hash.

Tree Use lexical order to insert into tree.

Red/Black Tree Tree balanced by colouring.

Splay Tree Moves frequently accessed to top.

Treap Tree balanced using random heap.

Tree is baseline (and subject to attack). Others are not (obviously) subject to attack.
Want flow lookup to:
- Should perform OK under typical traffic.
- Should not degrade badly under attack.
- Typical performance depends on keys.
- Collect trace of traffic,
- Assess using pcap framework,
- Check performance in kernel.
Big CPU

![Graph showing the average CPU per packet (us) for different hash table types and algorithms as a function of packets processed.]
Small CPU

Average CPU per packet (us)

Packets processed

Pearson (Byte) Hash Table
Xor (Byte) Hash Table
Treap
Unbalanced Tree
Red/Black Tree
Splay Tree
SHA Hash Table
MD5 Hash Table
Pearson (Word) Hash Table
Universal Hash Table
Xor (Word) Hash Table
Peak Forwarding

The graph shows the peak forwarding performance for different filter algorithms:
- No ipfw
- ipfw with Xor
- ipfw with Universal
- ipfw with Pearson

The x-axis represents the number of packets in, while the y-axis shows the number of packets out. Each line represents a different filter algorithm, allowing for a comparison of their performance under varying packet input loads.
Conclusion

- Take care choosing algorithms.
- XOR doesn’t look so good.
- Universal hash actually pretty cheap.
- Looking at attacking some hashes:
  - …Simple Pearson.
  - …RSS Hash on Ethernet Cards.