Network Security

George Varghese

January 13, 2005

Security Outline

Examples of security functions that can be implemented at high speed

- *17.1, String Searching in Packet Payloads:* hardware support for looking for multiple suspicious strings in packet payload.
- *17.2, Approximate String Matching:* hardware support for approximate matching of suspicious strings in packet payload.
- *17.3, IP Traceback via Packet Marking*: hardware support for tracing a packet's source using a few bits in each packet.
- *17.4, IP Traceback via Logs:* hardware support for logging packets efficiently for later use in traceback and other applications without using bits in packets.

17.1 String Searching in Packet Payloads

- Signature based Intrusion detection systems go beyond classifiers in allowing a 5-tuple rule *plus* an arbitrary string that can *appear anywhere in the packet payload*.
- Snort has 300 such augmented rules with 300 possible strings to search for. For example, an attempt to run the Perl interpreter can be caught by looking for string "perl.exe" in packet.
- Snort used to do string search by matching each packet against each Snort rule in turn. For each rule that matches in the classifier part, Snort will run a Boyer-Moore search on the corresponding string, potentially doing several string searches per packet.
- Can one search for all possible strings in one pass through packet?

Integrated String Matching using a Trie

b a b a b a r . . . (PACKET PAYLOAD)

not b from most nodes

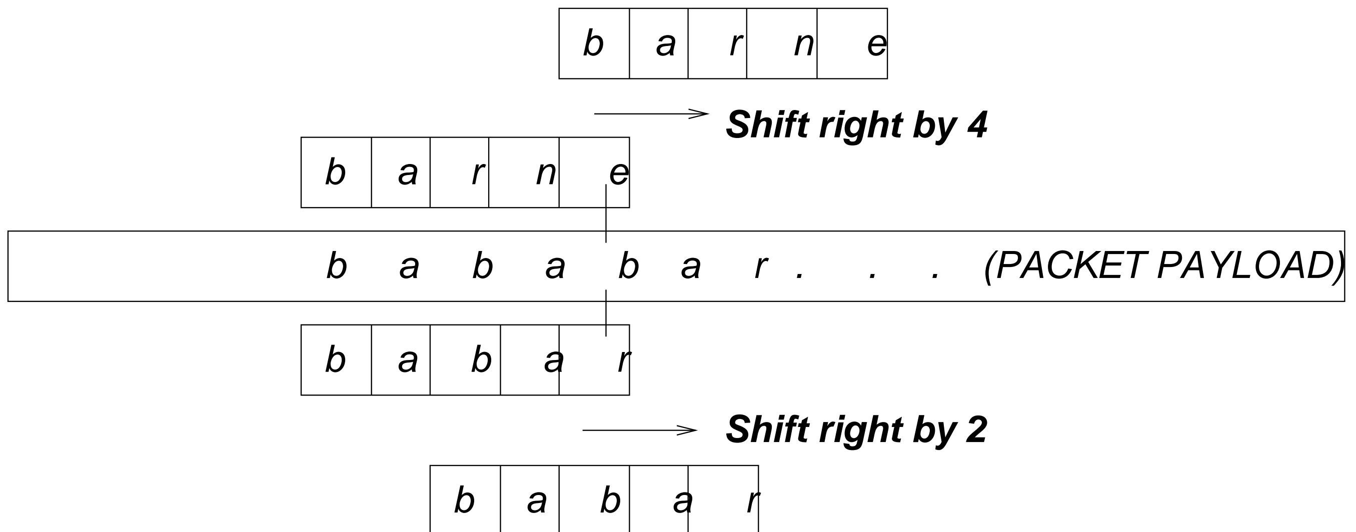
b from most other nodes

babar

barney

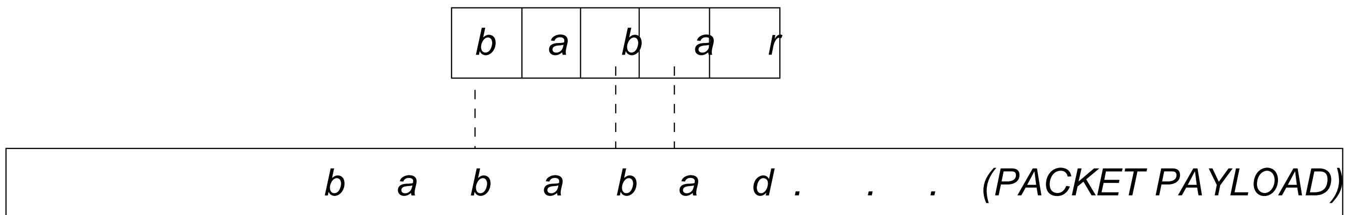
- Aho-Corasick is a trie with backpointers for fast failure backtracking.
- Follow characters in content starting with root, identify a string if leaf is reached. Fairly easy in hardware after trie is precomputed.

Integrated String Matching using Boyer-Moore



- Boyer-Moore faster on average because of large potential skipping. Use smallest skip across all strings S to align bad character with rightmost occurrence of character in string in S.
- Proposed concurrently by *Coit, Staniford, and McAlerney* and *Fisk and Varghese*. Latter implementation ported to Snort. Too inefficient to use only 1 set of all possible strings . . .

17.2 Approximate String Matching via Random Projections

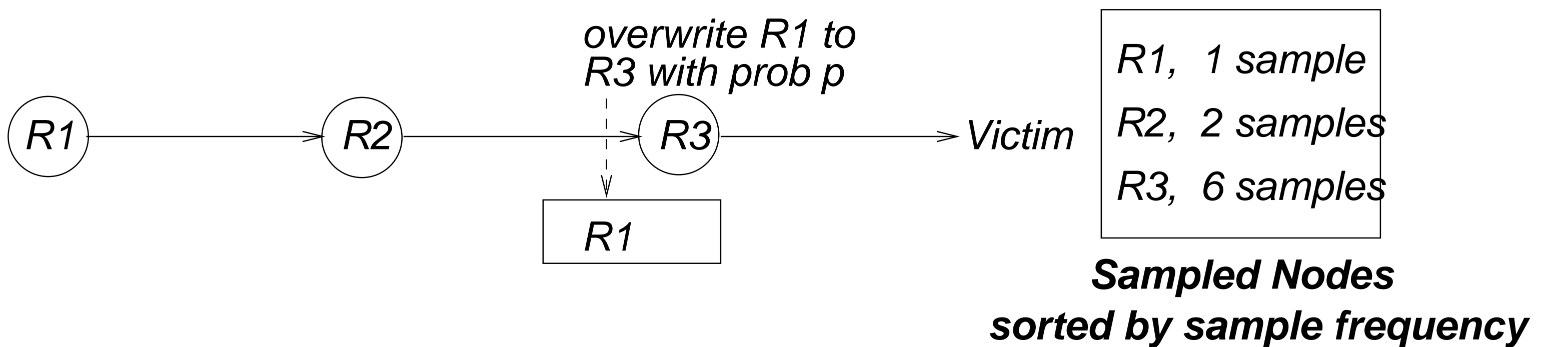


- Some security applications can benefit from approximate string searching in case intruder adds blanks etc.
- One idea is to search as before but only for a randomly selected subset of characters initially to filter out potential suspects. *Indyck-Motwani*.

17.3 IP Traceback via Probabilistic Marking

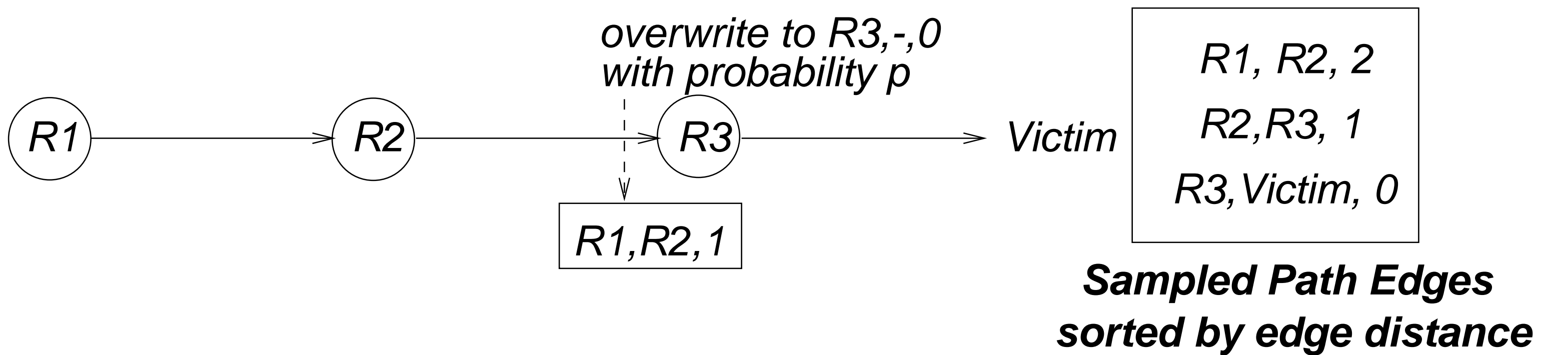
- **Setting:** Find a way to trace Denial-of-Service back to originating source point despite fake source addresses
- **Problem:** Easy to do by logging router path in packet header but very few extra header bits. Exploit fact that most attacks have many packets?

Traceback via Node Sampling



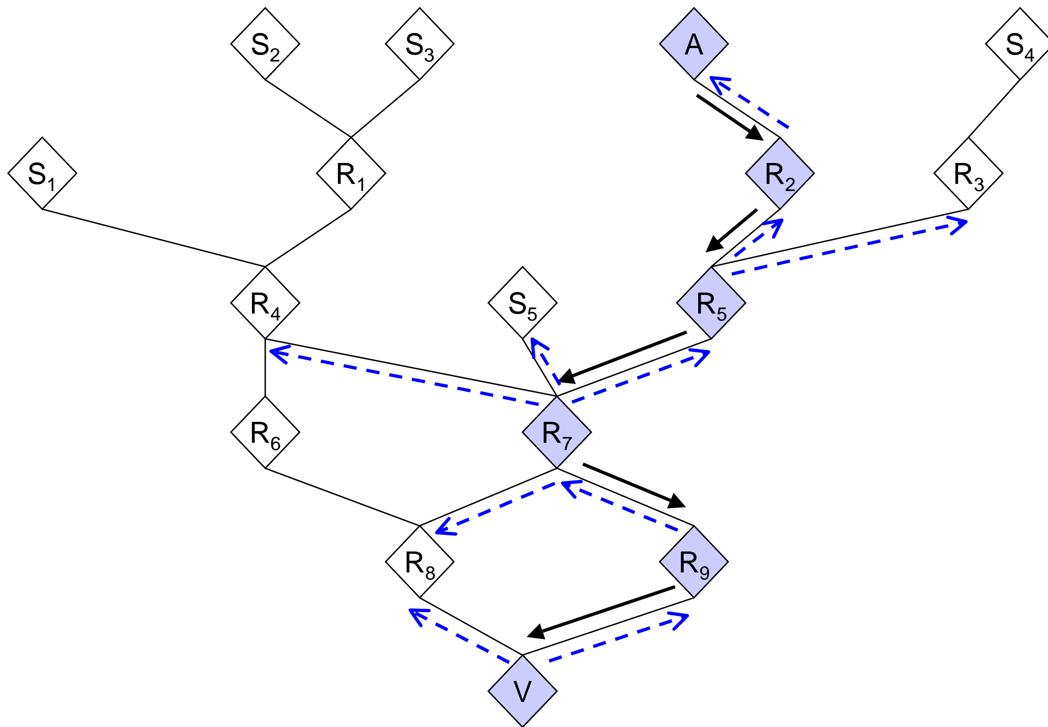
- Each router writes its ID into a *single* node ID field in packet with probability p.
- **Problem:** Many samples needed to deal with random variation in inferring order, and large time $p(1-p)^{D-1}$ to receive furthest node for large p as needed to foil malicious marks. 42,00 packets for $p = 0.51$ and $D = 15$.

IP Traceback via Edge Sampling (*Savage et al*)



- If sampled with prob p, write $(ID, -, 0)$; else, if $d = 0$, write ID to second field and increment d unconditionally.
- **Idea:** Explicit distance prevents extra samples to deal with random variation. Also, allows use of small $p \approx 1/D$ which makes $p(1-p)^{D-1} \approx ep/(1-p)$, small. $p = 1/25$. Further reduction: chop node IDs into fragments.

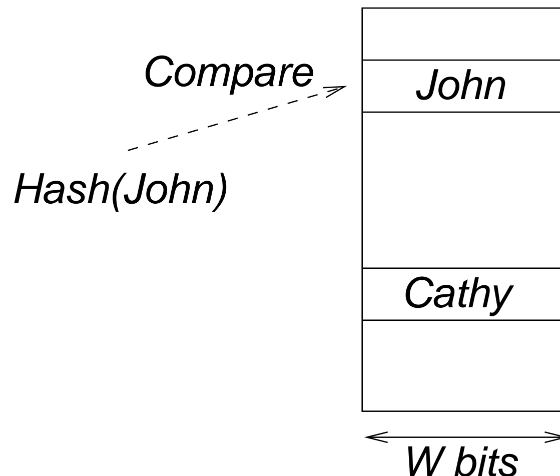
17.4 IP Traceback via Logging (*Snoeren et al*)



- **Problem:** Edge sampling requires changes to IP forwarding and does not work for single packet attacks like *Teardrop*.
- **Solution:** Log a 32 bit hash of invariant content of packet. Query routers on each link, starting from victim, to identify path. 32 bits per packet huge.

Towards Bloom Filters

ALLOWED USERS

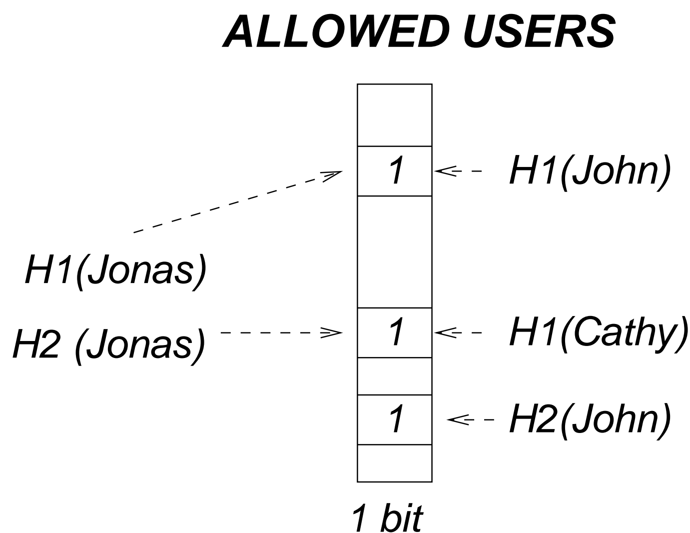


Is John an allowed user?

- Querying a packet log or a table of allowed users is a *set membership query*, often implemented by a hash table.

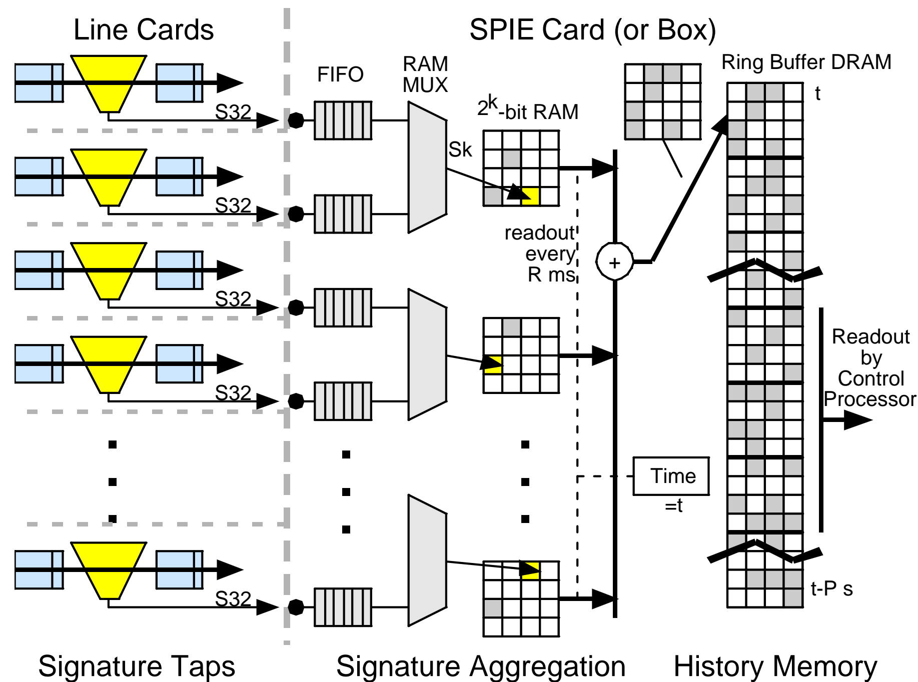
- Each hash table entry is $\geq W$ bits, where W is size of member identifier. Can reduce from W to say 5 by doing 3 more writes and by *relaxing specification*: accepting some probability of error (false positive).

Bloom Filter Idea



- False positive rate for m size bitmap to store n members using k hash functions is $\left(1-(1-1/m)^{kn}\right)^k \approx (1-e-kn/m)^k$.
- For $k = 3$ and 5 bits per member ($m/n = 5$), false positive rate is roughly 1 percent.

Bloom Filter Implementation of Packet Logging



- As in case of counters, use of SRAM for fast front-end *random access* allows speed. Bloom filter reduces from 32 to 5 bits per packet at cost of 3 SRAM read-modify writes.
- DRAM backing store can be written into and keep up with speed because of *sequential accesses* to DRAM log using a wide word (DRAM row).