Filtering Architectures

Olivier Paul
LOR department

Overview

- Introduction
- Network level Packet Filtering
  - Packet classification
  - Fragmentation
- Circuit level Packet Filtering
  - Stateful filtering
  - TCP stream reassembly
  - Directing traffic to filters
- Application level Packet Filtering
  - Protocol parsing
  - Pattern matching
  - Encryption
- Packet filter examples

A few references

- …That helped me setting up this course.
  - Real Stateful TCP Packet Filtering in IP Filter, Guido van Rooij, Usenix 01, 2001.
  - PF: http://www.benzedrine.cx/pf.html
  - NETFILTER: http://www.netfilter.org/

Network level Filter

- Reminder
Introduction

• What packet filtering does?
  – Provide a form of access control:
    • a security service used to protect resources against unauthorized use.
  – Decide if a “communication” is allowed according to packet content.
  – How to relate packet filtering to access control?
    • Resources implies a form of resources identification
      – Network services = resources.
    • Unauthorized implies some form of user identification
      – Network address = users.
    • Degree of trust in those addresses can change widely depending on the location of the filtering device.

• Packet filtering cons:
  – Communication authorization not based on strong authentication information.
  – IP address can be changed (User, NAT).
  – Little user related information.
    • One address = many users.
  – Little resource related information.
    • No strong link between service and port number.
    • Some port number are attributed dynamically.
    • (We’ll see more about this later).

• Packet filtering pros:
  – Simplicity.
    • Easy to manage.
      – Does not require authentication management or deployment in applications.
      – “Single” point of service.
    • Widely available (no standard interoperability problem) and cheap.
  – Fast.
    • Authentication: up to 1 Gbit/s.
    • Packet filtering: up to 100 Gbit/s.
  – Sole solution to some specific problems.
    • Only solution to discard information before authentication.

• Where can you find packet filters?
  – Security devices:
    • Firewalls.
    • VPN gateway.
  – Network devices.
    • Switches.
    • Routers.
    • ADSL/Cable modems.
  – End systems
    • Most UNIX’s.
    • Windows 2000+.
Network level Filter

- Reminder

Packet Classification

- Motivation
  - Differentiate traffic treatment
    - QoS related functionality
      - CAR, Marking, …
    - Policy based routing.
  - Security related functionality
    - Access control lists, VPN, …
  - Accounting and billing
    - Obtain Traffic sample
      - Flow based, Packet based, …

Classification

- Problem to solve
  - Using several fields in packet header

Classification

- Problem to solve
  - Using several fields in packet header ...
Classification

- Problem to solve
  - ... and a policy carrying on these fields ...
    - Example
      access-list 101 permit ip host 192.168.200.1 host 192.168.204
      access-list 101 permit ip host 192.168.203.1 host 192.168.203
  - ... Decide which action applies to packet.
    - Example: permit, deny, rate limit to 1Mb/s, mark yellow, encrypt ...

- Example
  - Decide which action applies to packet.

Introduction

Classification

• In a more formal way:
  Given a Policy P
  \[ P = \{ R_0, \ldots, R_i, \ldots, R_n \} \]
  \[ R_i = (O_i, C_i, A_i) \]
  \[ C_i = \{ C_i0, \ldots, C_ip, \ldots, C_ip \} \]
  \[ C_ip = (L_ip, U_ip) \]
  And given a Packet T
  \[ T = \{ V_0, \ldots, V_p \} \]
  Find \( R_k \in P \) such that \( \forall e \in [0..p], L_ke < V_e < U_ke \)
  And there is no \( k' \) such that \( O_k < O_{k'} \)

The good news is:

- Problem is related to computational geometry problem.
  Point location problem (2D version)

The bad news is:

- Problem does not have a nice solution in the general case
  - Best solution in term of temporal complexity:
    \[ SC \in O(n), TC \in O(n^2) \]
  - Best solution in term of spatial complexity:
    \[ SC \in O(n), TC \in O(n log n) \]

\[
\begin{array}{ccc}
  n & k & \text{Spatial Complexity} \\
  10 & 5 & 100000 \\
  100 & 5 & 10. E97 \\
  1000 & 5 & 1.E15 \\
\end{array}
\]

\[
\begin{array}{ccc}
  n & k & \text{Temporal Complexity} \\
  10 & 5 & 243 \\
  100 & 5 & 16807 \\
  1000 & 5 & 100000 \\
\end{array}
\]

- Overmars, Van der Stappen [Journal of Algorithms 96]
Classification

• Usual complexities
  – Firewall:
    • 5-6 fields.
    • 2k rules.
  – Microflows (e.g. RSVP - edge):
    • 4 fields.
    • 128k-1M rules.
  – Filtering router:
    • 5 fields.
    • 16-128k rules.

Address lookup

• Interesting question.
  – Is the lookup problem a sub-problem of packet classification?

• Answer is yes.
  – Address A and Netmask M can be coded as a range
    • \( C_i = (L_i, U_i) \), \( L_i = (A \text{ AND } M) \), \( U_i = ((A \text{ AND } M) \text{ AND } \text{NOT } M) \)
  – Prefix length can be coded as priority.
    • \( \text{Mask}(A) \geq \text{Mask}(A) \iff O(A) < O(A) \)
  – Action can be coded as output port.

Packet Classification

• Practical methods
  – Linear search.
  – Ternary CAM.

• Other methods
  – Lookup methods extensions.
    • Grid of tries [Srin99]
    • Fat Inverted Segment Tree [Fel00]
  – Parallelization:
    • Bitmap intersection [Lak98]
  – Heuristics:
    • Recursive Flow classification [Gup99]
    • Hierarchical Intelligent Cuttings [Gup99]
  – General improvements:
    • Heap on trie [Gup00]
    • Specific instances of the problem.
    • Many more...
Packet Classification

- Practical methods
  - Linear search.
  - Ternary CAM.

- Other methods
  - Lookup methods extensions.
    - Grid of tries [Srin99]
    - Fat Inverted Segment Tree [Fel00]
  - Parallelization:
    - Bitmap intersection [Lak98]
  - Heuristics:
    - Recursive Flow classification [Gup99]
    - Hierarchical Intelligent Cuttings [Gup99]
  - General improvements:
    - Heap on trie [Gup00]
  - Specific instances of the problem.
  - Many more...

Packet Classification

Linear Search

If @S=A and @D=C and SP=C and DP=D then permit
If @S=A and @D=B and SP=C and DP=E then deny
If @S=A and @D=B then permit
If @S=A and @D=C then deny

Flow

A

C

C

C

Deny

• CT: O(n d), CS: O(n d), insertion O(1).

Content Addressable Memory

- CAM
  - Extension of lookup answer.

• Internal structure
  - Memory content
  - OR
  - Encode
  - Location
  - CAM
  - match/no match

Content Addressable Memory
**Ternary CAM**

- Ternary CAM behaviour

```
xbits bits pattern
```

```
TCAM
```

```
Address
```

- x bits pattern Policy
- Allowing joker bits.
- 1?000?10?111?0?

**CAM advantages:**
- Fast: 6-10ns access time.
- Simple.
- Multi-fields.

**CAM limitations:**
- Chip complexity: Grows linearly with number of entries.
- Power consumption: 7-15w/MB (SRAM 0.25w/MB)
- Size: 2MB (SRAM ~8-16MB)
- Cost: 100$/MB (SRAM ~5$/MB)
- Number of entries (Size/pattern size)
- Pattern size (0~500 bits)
Packet Classification

• Practical methods
  – Linear search.
  – Ternary CAM.

• Practical architectures
  – Caching.

• Other methods
  – Lookup methods extensions.
    • Grid of tries [Sin99]
    • Fat Inverted Segment Tree [Fel00]
  – Parallelization:
    • Bitmap intersection [Lak98]
  – Heuristics:
    • Recursive Flow classification [Gup99]
    • Hierarchical Intelligent Cuttings [Gup99]
  – General improvements:
    • Heap on trie [Gup00]
    • Specific instances of the problem.
    • Many more...

Introduction
Application
Architecture
Network
Circuit
Caching

Caching

• Why does it work?
  – Temporal redundancies:
    • Packets often belong to a flow
      – Set of packet with similar characteristics.
        • (Source, Destination) x (Address, Port).
      • Knowing which action applies to the first packet provides the answer to any following packet with same characteristics.
    – In practice, filtering devices often see the same communications:
      • Ex: permit tcp src-ip our_net dst-ip any src-port >=1024 dst-port 80
      1. Often apply the same rules.
      2. Rules often apply to the same fields.

• What type of cache?
  – Content:
    • Rules: (packet classification algorithm unchanged).
      – Latest rules are kept in faster memory.
        » Registers, L1 cache, L2 cache…
      • Instantiated rules: (packet classification is changed).
        – Latest connection are kept in faster memory.
  – Size.
  – Update algorithm.
    • LRU.
Packet Classification

- **Practical methods**
  - Linear search.
  - Ternary CAM.

- **Other methods**
  - Lookup methods extensions.
    - Grid of tries [Srin99]
    - Fat Inverted Segment Tree [Fel00]
  - Parallelization:
    - Bitmap intersection [Lak98]
  - Heuristics:
    - Recursive Flow classification [Gup99]
    - Hierarchical Intelligent Cuttings [Gup99]
  - General improvements:
    - Heap on trie [Gup00]
  - Many more...

- **Practical architectures**
  - Caching.

Multi-dimensional radix tries

- **Multidimensional tries.**
  - Extension to binary tries.
    - Pb: In fields 1, 2, … we may have overlap between intervals.
    - Example

  - 101* ⊂ 10* ⊂ 1*

- **Superset propagation.**
  - If \( I_{k} \subset I_{j} \) then create \( T_{j+1} \) using \( I_{j, k+1} \) and \( I_{j, k+1} \)
  - Last match is the good one.

Ex: 101, 000

<table>
<thead>
<tr>
<th>Rule</th>
<th>Source Addr</th>
<th>Dest. Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>101*</td>
<td>01*</td>
</tr>
<tr>
<td>R2</td>
<td>10*</td>
<td>01*</td>
</tr>
<tr>
<td>R3</td>
<td>1*</td>
<td>00*</td>
</tr>
<tr>
<td>R4</td>
<td>01*</td>
<td>111*</td>
</tr>
</tbody>
</table>
Multi-dimensional radix tries

- Backtracking

<table>
<thead>
<tr>
<th>Rule</th>
<th>Source Addr</th>
<th>Dest. Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>101*</td>
<td>01*</td>
</tr>
<tr>
<td>R2</td>
<td>10*</td>
<td>10*</td>
</tr>
<tr>
<td>R3</td>
<td>1*</td>
<td>0*</td>
</tr>
<tr>
<td>R4</td>
<td>01*</td>
<td>111*</td>
</tr>
</tbody>
</table>

Ex: 101, 000

Depth first search and backtrack in the first trie if no match to explore every possibility
Cost: O(w^2)

Grid of Tries

- Two dimensional structure.
  - Idea: Use shortcut to prevent backtracking.

Ex: 101, 000

Using shortcuts
Time Complexity O(w)
Memory Complexity O(N \cdot w)

Would that work for larger dimensions?

Packet Classification

- Practical methods
  - Linear search.
  - Ternary CAM.
- Practical architectures
  - Caching.

- Other methods
  - Lookup methods extensions.
    - Grid of tries [Srin99]
    - Fat Inverted Segment Tree [Fel00]
  - Parallelization:
    - Bitmap intersection [Lak98]
  - Heuristics:
    - Recursive Flow classification [Gup99]
    - Hierarchical Intelligent Cuttings [Gup99]
  - General improvements:
    - Heap on trie [Gup00]
    - Specific instances of the problem.
    - Many more...

Bitmap Intersection

- Ex: Laksham, Stiliadis [SIGCOMM98], TC : O(log(2n+1)), SC : O(d.n^2).
**Bitmap Intersection**

- Optimizing bitmap storage.
  - $2N+1$ changes in the bitmap in each dimension at most.
  - Usually 1 bit change between 2 adjacent intervals.
  - Idea:
    - Store original bitmap and then the location of the bit that changed.
    - Bit changed can be stored on $\log(N)$ bits (since the size of each bitmap is $N$).
  - Example:

<table>
<thead>
<tr>
<th>Original</th>
<th>000</th>
<th>001</th>
<th>101</th>
<th>111</th>
<th>011</th>
<th>010</th>
<th>000</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>0</td>
<td>0</td>
<td>01</td>
<td>11</td>
<td>10</td>
<td>11</td>
<td>01</td>
</tr>
</tbody>
</table>

**Packet Classification**

- Practical methods
  - Linear search.
  - Ternary CAM.

**Recursive Flow Classification**

- Gupta, McKeown [SIGCOMM99]

- Other methods
  - Lookup methods extensions.
    - Grid of tries [Srin99]
    - Fat Inverted Segment Tree [Fel00]
  - Parallelization:
    - Bitmap intersection [Lak98]
  - Heuristics:
    - Recursive Flow classification [Gup99]
    - Hierarchical Intelligent Cuttings [Gup99]
  - General improvements:
    - Heap on trie [Gup00]
    - Specific instances of the problem.
    - Many more...

- TC : $O(1)$, SC : $?$. Insertion 10s
Network level Filter

- Reminder

Fragmentation Management

- Without state maintenance.
  - RFC 1858.
  - RFC 3128.
- With state maintenance.
  - With reassembly
    - RFC 791.
    - RFC 815.
  - Without reassembly

Fragmentation Management

- RFC 3128 (eg: most routers).
  - Filters packets depending on the packet characteristics without keeping a state.
  - IP packets with less than 8 bytes of data have to be dropped.
    - 8 bytes is the minimal size in RFC 791.
    - Prevents inability to classify first IP packet due to lack of information.
  - TCP packets with 8 bytes of data have to be dropped.
    - Prevents packets without TCP flag information.
  - TCP packets with offset=1 (8 bytes offset).
    - Prevents flag information to be reassembled differently between packet filter and end host.

- RFC 3128 (eg: most routers).
  - Pro:
    - Cheap: Does not necessitate state maintenance in the packet filter.
    - Efficient: Insures that transport level information is considered in a non ambiguous way.
  - Cons:
    - Based on the assumption that end host will drop packets when the transport information cannot be obtained.
      - Some packets are not filtered.
        - Ex: Send only fragments with Offset>1, Data size > 8 and no transport information.
      - Some attacks can take advantage vulnerable reassembly mechanisms:
        - DoS attacks on memory.
        - Buffer overflows using offset larger than 65kBytes.
Fragmentation Management

- RFC 3128 (eg: most routers).
  - Cons:
    - Application level information can still be reassembled ambiguously.

Windows NT  BSD

Fragmentation Management

- With reassembly (eg: pf), RFC 815.
  - Reassemble context lookup.

<table>
<thead>
<tr>
<th>[Src_addr, Dst_addr, Proto, ID]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Hole->first = 512
Hole->last = 1024
Hole->previous
Hole->next

Fragments can be stored in buffer
( fragment size > Hole struct)

For (hole = hole_head; hole != NULL; hole = hole->next) {
  if (frag->first > hole->last) continue;
  if (frag->last < hole->first) continue;
  hole_tmp = new (HOLE);
  if (frag->first > hole->first) {
    hole_tmp->first = hole->first;
    hole_tmp->last = frag->last - 1;
  }
  if (frag->last < hole->last) {
    hole_tmp->first = fragment->last + 1;
    hole_tmp->last = hole->last;
  }
  attach(hole_tmp); free(hole);
}
**Fragmentation Management**

- **Without reassembly (eg: ipfilter)**
  - Drop fragments as long as no transport level information is in the cache.
  - Drop fragments that do not comply with RFC 3128.
  - When receiving fragment with full transport level information store transport information and fragmentation information in the cache.
  - Fragments are then matched to the content of the cache.
    - Verify that fragments do not overlap – crop/drop overlapping fragments.
    - Verify that fragments are not duplicated – drop duplicates.
    - Verify that fragments will not take advantage of known vulnerabilities.

**Pros**
- Prevents some forms of attack.
- More resilient to DoS attacks.

**Cons**
- Does not insure a non ambiguous reassembly at the end (depends on amount of cropping done in the filter).
- Assumes reordering is always illegitimate.
  - Packets can be reordered naturally.
    » Packets follow different paths in the network.
    » Buggy IP stacks (e.g. linux 2.4) send fragments in reverse order.

**Circuit level Filter**

- **Reminder**

**Circuit Level Filters**

- Two main types of architectures:
  - Circuit level proxy.
    - Usually implemented in user space.
    - Proxy acts as a server for the client, as a client for the server.
    - Usually not transparent.
  - Stateful packet filter.
    - Extension to packet filters, implemented in the kernel.
    - Mostly Transparent.
  - Similarities:
    - Try to avoid some attacks
      - Packets/segment insertion/evasion.
      - Fingerprinting.
        - by using transport level information.
        - and a state
    - Can bind user with connection if associated with authentication.
Circuit Level Filters

- State location
  - Circuit level proxy
  - Stateful packet filter

Stateful packet filtering

- State checking
  - What type of state are we talking about?
    - Connection exists between hosts.
    - State automaton.
      - Maintain a state for each side.
      - Reject packets that do not satisfy TCP state automaton transitions.
Stateful packet filtering

• State checking
  – What type of state are we talking about?
    • TCP Initial Sequence number evolution.
      – Common practice up to 1995: RFC 793
        » Increase ISN by one every 4 microseconds.
      – Alternative practice: 4.4BSD
        » Increase ISN by 64000 every 1/2 second.
    • 1995: Steve Bellovin paper about connection hijacking.
      – If ISN is easy to guess it is easy to find segments that can be accepted by the receiver (and a firewall).
      – If ISN follows a well defined evolution pattern you can infer new ISN from older ones.
    • “Most” systems now use RNG to generate ISN.
    • However slightly increases the chance to have old packets accepted.

---

Stateful packet filtering

• State checking
  – What type of state are we talking about?
    • TCP Sequence number guessing.

---

Stateful packet filtering

• State checking
  – What type of state are we talking about?
    • Filtering process: Source side.

---

Stateful packet filtering

• State checking
  – What type of state are we talking about?
    • Filtering process: Source side.
Stateful packet filtering

- State checking
  - What type of state are we talking about?
  - Filtering process: Destination side.

\[
\begin{align*}
\text{Data:} & \quad s_n + s_d = \text{max} (\text{ack}_s + n_d) \\
\text{Ack:} & \quad s_n = \text{max} (\text{ack}_s - n_d) \\
& \quad \text{ack}_s = \text{max} (s_n + n_d) - \text{MAXWIN}
\end{align*}
\]

Stateful packet filtering

- State checking
  - What type of state are we talking about?
  - What about connectionless protocols (UDP, ICMP)?
    - Automation approach still applies.
    - Timers play a bigger role to match traffic to an existing communication.
      - "Connection" timeout.

Circuit level Filter

- Reminder

\[
\begin{align*}
\text{Incoming Packets} & \rightarrow \text{Internal Interface} \\
& \rightarrow \text{Filter} \\
& \rightarrow \text{External Interface} \\
& \rightarrow \text{Authorized Packets}
\end{align*}
\]
**Stateful packet filtering**

- State/Rule classification relations

  ![Diagram](image)

- State lookup
  - An opportunity to improve performance

  - Popular data structures:
    - Trees: balanced tree (e.g. pf), Splay trees (e.g. stream4).
      - Average/worst complexity: Search $O(\log n)$, update $O(\log n)$.
    - Linked Lists.
      - Average/worst complexity: Search $O(n)$, update $O(1)$.
    - Hashing (e.g. ipfilter).
      - Average complexity: Search $O(1)$, update $O(1)$.

**Hashing**

- Hash function + SRAM.

  ![Diagram](image)

- Hash function limitations

  ![Diagram](image)

  Since $n < \text{rule size}$ we can get some collisions!
  You need to linearly test every rule in the collision queue!
Hashing

- Usual tricks with hash functions
  - Use several functions

Class. Rule 1
(@S, @D, SP, DP)

Class. Rule 2
(@S', @D', SP', DP')

Policy storage

Fill the queue with the less elements

- Andrei Broder, Michael Mitzenmacher, INFOCOM 2000

Stateful packet filtering

- State lookup
  - Question:
    - Can we limit ourselves to searching the connection matching (src addr, src port, dst addr, dst port)?
  - First problem: Returning packets.
    - Carry (dst addr, dst port, src addr, src port)

- Second problem:
  - We would like to be able to accept the returning ICMP if it can be associated with an outgoing connection.
  - Need to parse states that may not be directly related to incoming packets.
  - Should work with other problems
    - ICMP network unreachable (source different).
    - FTP (Protocol different).
Circuit level Filter

• Reminder


Normalization

• IP Level

<table>
<thead>
<tr>
<th>#</th>
<th>IP Field</th>
<th>Normalization Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Version</td>
<td>Non-IPv4 packets dropped.</td>
</tr>
<tr>
<td>2</td>
<td>Header Len</td>
<td>Drop if hdr len too small.</td>
</tr>
<tr>
<td>3</td>
<td>Header Len</td>
<td>Drop if hdr len too large.</td>
</tr>
<tr>
<td>4</td>
<td>Diffserv</td>
<td>Clear field.</td>
</tr>
<tr>
<td>5</td>
<td>ECT</td>
<td>Clear field.</td>
</tr>
<tr>
<td>6</td>
<td>Total Len</td>
<td>Drop if tot len link layer len.</td>
</tr>
<tr>
<td>7</td>
<td>Total Len</td>
<td>Trim if tot len link layer len.</td>
</tr>
<tr>
<td>8</td>
<td>IP Identifier</td>
<td>Encrypt ID.</td>
</tr>
<tr>
<td>9</td>
<td>specific protocols</td>
<td>Protocol pass packet to TCP, UDP, ICMP handlers.</td>
</tr>
<tr>
<td>10</td>
<td>Frag offset</td>
<td>Reassemble fragmented packets.</td>
</tr>
<tr>
<td>11</td>
<td>Frag offset</td>
<td>Drop if offset + len &gt; 64KB.</td>
</tr>
<tr>
<td>12</td>
<td>DF</td>
<td>Clear DF.</td>
</tr>
<tr>
<td>13</td>
<td>DF</td>
<td>Drop if DF set and offset &gt; 0.</td>
</tr>
<tr>
<td>14</td>
<td>Zero flag</td>
<td>Clear.</td>
</tr>
<tr>
<td>15</td>
<td>Src addr</td>
<td>Drop if class C or E.</td>
</tr>
<tr>
<td>16</td>
<td>Src addr</td>
<td>Drop if MBytes=127 or 0.</td>
</tr>
<tr>
<td>17</td>
<td>Src addr</td>
<td>Drop if 255.255.255.255.</td>
</tr>
<tr>
<td>18</td>
<td>Dst addr</td>
<td>Drop if class E.</td>
</tr>
<tr>
<td>19</td>
<td>Dst addr</td>
<td>Drop if MBytes=127 or 0.</td>
</tr>
<tr>
<td>20</td>
<td>Dst addr</td>
<td>Drop if 255.255.255.255.</td>
</tr>
<tr>
<td>21</td>
<td>TTL</td>
<td>Raise TTL to configured value.</td>
</tr>
<tr>
<td>22</td>
<td>Checksum</td>
<td>Verify, drop if incorrect.</td>
</tr>
<tr>
<td>23</td>
<td>IF options</td>
<td>Remove IF options.</td>
</tr>
<tr>
<td>24</td>
<td>IF options</td>
<td>Zero padding bytes.</td>
</tr>
</tbody>
</table>

• ICMP Level

<table>
<thead>
<tr>
<th>#</th>
<th>ICMP Type</th>
<th>Normalization Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Echo request</td>
<td>Optionally drop if ping checksum incorrect.</td>
</tr>
<tr>
<td>3</td>
<td>Echo request</td>
<td>Zero “code” field.</td>
</tr>
<tr>
<td>4</td>
<td>Echo request</td>
<td>Optionally drop if ping checksum incorrect.</td>
</tr>
<tr>
<td>6</td>
<td>Echo reply</td>
<td>Drop if no matching request.</td>
</tr>
<tr>
<td>7</td>
<td>Source quench</td>
<td>Optionally drop to prevent DoS.</td>
</tr>
<tr>
<td>8</td>
<td>Destination Unr.</td>
<td>Unscramble embedded scrambled IF identifier.</td>
</tr>
</tbody>
</table>

• TCP Level

<table>
<thead>
<tr>
<th>#</th>
<th>TCP Field</th>
<th>Normalization Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Seq Num</td>
<td>Enforce data consistency in retransmitted segments.</td>
</tr>
<tr>
<td>2</td>
<td>Seq Num</td>
<td>Trim data to window.</td>
</tr>
<tr>
<td>3</td>
<td>Seq Num</td>
<td>Cold-start; trim to keep-alive.</td>
</tr>
<tr>
<td>4</td>
<td>Ack Num</td>
<td>Drop ACK above sequence hole.</td>
</tr>
<tr>
<td>5</td>
<td>SYN</td>
<td>Remove data if SYN=1.</td>
</tr>
<tr>
<td>6</td>
<td>SYN</td>
<td>If SYN=1 &amp; FIN=1, drop.</td>
</tr>
<tr>
<td>7</td>
<td>SYN</td>
<td>If SYN=1 &amp; FIN=1, clear FIN.</td>
</tr>
<tr>
<td>8</td>
<td>SYN</td>
<td>If SYN=0 &amp; ACK=0 &amp; RST=0, drop.</td>
</tr>
<tr>
<td>9</td>
<td>RST</td>
<td>Remove data if RST=1.</td>
</tr>
<tr>
<td>10</td>
<td>RST</td>
<td>Make RST reliable.</td>
</tr>
<tr>
<td>11</td>
<td>RST</td>
<td>Drop if not in window.</td>
</tr>
<tr>
<td>12</td>
<td>FIN</td>
<td>If FIN=1 &amp; ACK=0, drop.</td>
</tr>
<tr>
<td>13</td>
<td>PSH</td>
<td>If PSH=1 &amp; ACK=0, drop.</td>
</tr>
<tr>
<td>14</td>
<td>Header Len</td>
<td>Drop if less than 5.</td>
</tr>
<tr>
<td>15</td>
<td>Header Len</td>
<td>Drop if beyond end of packet.</td>
</tr>
<tr>
<td>16</td>
<td>Window</td>
<td>Remove window withdrawals.</td>
</tr>
<tr>
<td>20</td>
<td>Checksum</td>
<td>Verify, drop if incorrect.</td>
</tr>
<tr>
<td>23</td>
<td>URG</td>
<td>If URG=1 &amp; ACK=0, drop.</td>
</tr>
<tr>
<td>24</td>
<td>MSS option</td>
<td>If SYM=0, remove option.</td>
</tr>
<tr>
<td>25</td>
<td>MSS option</td>
<td>Cache option, trim data to MSS.</td>
</tr>
<tr>
<td>26</td>
<td>WS option</td>
<td>If SYM=0, remove option.</td>
</tr>
</tbody>
</table>

We’ll look at this one on the next slides.
TCP Reassembly Problems

- Retransmissions – Classic timeout

A \( (x-x+s) \)

B \( (x+s-x+2s) \)

C \( (x+2s-x+3s) \)

 Ack \( x+s \)

B \( (x+s-x+2s) \)

 Ack \( x+s \)

1 duplicate Ack
No timeout
No Loss

TCP Reassembly Problems

- Retransmissions – Fast retransmit

A \( (x-x+s) \)

B \( (x+s-x+2s) \)

C \( (x+2s-x+3s) \)

D \( (x+3s-x+4s) \)

E \( (x+4s-x+5s) \)

 Ack \( x+s \)

 Ack \( x+s \)

 Ack \( x+s \)

 Ack \( x+s \)

3 duplicates Acks
No timeout

TCP Reassembly Problems

- Retransmissions – Fast retransmit – why 3 ?

A \( (x-x+s) \)

B \( (x+s-x+2s) \)

C \( (x+2s-x+3s) \)

D \( (x+3s-x+4s) \)

E \( (x+4s-x+5s) \)

 Ack \( x+s \)

 Ack \( x+s \)

 Ack \( x+s \)

 Ack \( x+s \)

3 duplicates Acks
No timeout

TCP Reassembly Problems

- Can we remove all ambiguities ?
  - Retransmission/Overlap alternatives:
    - Discard data that has already been seen.
      - Pd: If data is lost between filter and destination, retransmission is dropped.
      - Pd: Transmission is no longer reliable.
    - You can guess the need for a retransmission by looking at 3 duplicate acks and validate segments carrying ack value.
      - Pd: Does not work on retransmission with classic timeout.
    - Discard data that has already been acknowledged.
      - Pd: If ack is lost after filter, sender retransmits at timeout.
      - Risk to have connection total timeout + reset.
      - Pd: Sender can simultaneously send several time the same segment with different content.
      - Receiver can still reassemble data differently from filter.
      - Does not resolve ambiguities !!
TCP Reassembly Problems

- Can we remove all ambiguities and be non blocking?
  - Not really !
    - But we can remove some of them.
  - We have the same problem for any stateful protocol.
- Answers:
  - Use proxy like architecture where traffic is acknowledged locally.
  - Use knowledge about the destination to reassemble its way. (active mapping or configuration). (Not sufficient)
  - Use additional traffic between filter and destination to know state:
    - Send additional RST to make sure RST arrived at destination.
    - Use TCP keepalive to know if data packet was received.
    - not sufficient – (e.g. URG data problem).
  - Rewrite TCP segments to ensure unambiguous retransmission.
    - Very costly and not sufficient – (e.g. URG data problem).
  - Generate alerts on suspicious events.
    - Overlaps are frequent but overlaps with different content aren’t.

TCP Stream Reassembly

- Enforce Data consistency
  - Problem somehow similar to packets reassembly.
  - But segmentation problem much more frequent.
    - Fragmented traffic: ~0.5% Shannon & al. ACM TON 2002.
    - Segmented traffic: Any TCP traffic (75-80% existing traffic).
  - Duplicate fragments are not supposed to exist.
  - Is TCP reassembly costly ?
    - Number of states. Depends on:
      - Number of concurrent connections.
      - Average number of holes per connection.
      - Size of holes.
      - Holes duration.
  - How do holes appear ?
    - Packet Loss.
    - Reordering (routes changes/parallel path network processors).
    - Attacks.

TCP Reassembly cost

- Dharmapurikar & Paxson, USENIX Security 2005:
  - 3-20% connections include holes.
  - 95-97% connections include one hole.
TCP Reassembly cost

- Dharmapurikar and Paxson, USENIX Security 2005:
  - Possible attacks:
    - Generate multiple holes per connection.
    - Limit number of concurrent holes per connection.
    - Generate multiple connections with reduced number of holes:
      - Limit number of connections with hole per IP address.
      - Store data once connection is established.
    - Generate reduced number of connections with holes per IP address.
      - Reclaim memory by freeing random connection state.
  - Cost (limiting possibility of attacks):
    - 30kB-120kB/connection.

Circuit level Filter

- Reminder

Circuit level Filters Position

- Proxy

- Stateful Packet Filter

Proxy specific problems

- Directing flows to the proxy
  - Manual configuration
    - Administrator or user specifies proxy address and port to application.
  - Automated Configuration
    - Mostly for browsers.
    - Using configuration scripts
      - Proxy Auto-Config
      - Web Proxy Autodiscovery Protocol.
  - Transparent proxying
Proxy specific problems

- Proxy Auto-Config
  - Proxy.pac file (independent from browser)
  - Javascript program providing proxy address depending on destination address.
  - Obtained from web server provided through browser configuration.
  - You still need to configure browsers.

1. Administrator/User configures browser with configuration server address.
2. When browser is launched configuration script is obtained from server.
3. Script is cached and used for each request.
4. For every request script is executed and provides proxy address.
5. Request is sent to the proxy.

Introduction

- Web Proxy Auto discovery Protocol.
  - Automates web server configuration address retrieval.
  - Using DHCP (preferred)
    1. DHCP Discover with Tag #252 (WPAD auto-proxy-config)
    2. DHCP Offer with Tag #252.
    3. DHCP Request or DHCP Inform if DHCP Information already obtained.
    4. DHCP Ack with Tag #252 and associated URL.
    5. Use DNS to convert URL to IP address.
  - Using DNS
    - Queries wpad.localdomain.name from leaf to root.
    - wpad.rc105.lor.int-evry.fr/proxy.pac
    - wpad.lor.int-evry.fr
    - wpad.int-evry.fr
    - Many associated security issues.

- Directing flows from the proxy
  - Packets sent to the proxy.
  - Proxy must obtain final destination address/port.
    - Fixed proxy configuration.
    - Proxy sets up connection to predefined destination.
    - Works well for dedicated reverse proxies (one proxy per service).
  - Information passed by the client
    - In-band transport:
      - Protocol naturally carries duplicate information.
      - E.g. HTTP
    - Out of band transport:
      - Separate protocol used to carry final destination information.
      - E.g. SOCKS
      - Transparent to user (except socks library configuration).

- SOCKS commands:
  - Establish TCP connection.
  - Request Port binding.
  - SOCKS responses:
    - Request Granted
    - Request Rejected/Failed
Proxy specific problems

- Directing flows from the proxy
  - In band transport.
  - Information passed by the client
    - Extended protocol used to carry final destination information.
    - Eg. FTP Proxy
    - Not Transparent to user.

User View

```
ftp> open 157.159.100.21 2121
Connected to 157.159.100.21.
220 server ready - login please
Name (157.159.100.21:paul_o): ftp@ftp.int-evry.fr
331 password required
Password: 230 login accepted
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ...
```

Server Logs

```
```

Proxy specific problems

- Transparency
  - Ability to direct packets to a proxy without knowledge from clients.
  - Incoming packets carry true destination address.
  - Simplifies end systems configuration.
  - Does not apply to reverse proxying.
  - Also known as intercepting proxy.
  - Two alternatives:
    - Transparency through NAT.
    - Transparency through internal redirection.

Introduction

- Network
- Circuit
- Application
- Architecture

Proxy specific problems

- Transparency through NAT
  1. Client IP layer routes packet toward destination.
  2. NAT. Dst Address, Dst Port -> Proxy Address, Proxy port
  3. IP layer routes packet toward proxy device.
  4. Not much
  5. Proxy must obtain destination address to establish connection
     1. In band redundant destination information (e.g. HTTP)
     2. Dialog with device carrying NAT.
  6. NAT. Proxy Address, Proxy Port -> Svc Address, Svc Port

Proxy specific problems

- Transparency through internal redirection (alternative).
  1. Client IP layer routes packet toward destination. Proxy is bound to special socket (e.g. divert).
  3. IP layer sends packet to special socket identified by mark. (Not the traditional IP behavior)

Application level Filter

- Overview

Dealing with encryption

- SSL Man in The Middle proxy
  1. TCP Connections with proxy
     1. TCP handshake between client and proxy.
     2. TCP handshake between proxy and destination.
  3. SSL/TLS Negotiations
    1. Proxy gets certificate from destination server.
    2. Proxy validates destination server certificate (uses cache as speed up).
    3. Proxy negotiates shared key key_proxy_server with destination server.
    4. Proxy provides locally generated certificate to client.
    5. Client accepts certificate.
    6. Proxy negotiates shared key key_client_proxy with client.
Dealing with encryption

- SSL Man in The Middle proxy

1. Client sends traffic to proxy.
2. Proxy decrypts traffic using key\textsubscript{client-proxy}.
3. Eventually changes traffic content to hide that proxy is used.
   - E.g. HTTP Proxy Connection: Keep-Alive \rightarrow Connection: Keep-Alive
4. Proxy encrypts traffic using key\textsubscript{proxy-server}.

Application level Filter

- Overview

Protocol Analysis: Two kinds of problems:
- Given a flow of data determine which protocol is used (traffic recognition).
- Given a flow of data determine the structure and value of the information exchanged (message parsing).
Traffic recognition

- Recent field of research due to high amount of obfuscation used by some applications (e.g. P2P).
- Existing approaches:
  - Port number – application mapping.
  - Matching well known patterns associated with popular protocols.
  - Using heuristics to separate different traffic classes.

Pattern Matching

- From: protocol specifications/ protocol exchanges extract patterns characterizing protocol.
  - Eg: Kazaa (from l7-filter):
    - "get /.download/*|--*/.supernode|--|--*/.status|--|--*/.files|--|--*/.hash=[0-9a-f]*|--|--*/
    - http/1.1[user-agent: kazaa|x-kazaa(-username|-network|-ip|-supernodeIp|-xferid|-xferuid|tag]|"give [0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]"
  - Maintains connections states.
  - Examines packets independently.
  - Once a packet matches the pattern, associate application tag with connection.

Pattern Matching

- Advantages:
  - Maintains connections states.
  - Examines packets independently.
  - Once a packet matches the pattern, associate application tag with connection.

- Drawbacks:
    - Less than 10Mbs for l7-filter on 750Mhz P3.
    - False negative (source : (1)): 1-10% depending on protocol.
Pattern Matching

- Open Source Tools examples:
  - Mostly under Linux.
  - ~70 signatures. Many with false positives/negatives.
  - Mostly P2P oriented. ~10 signatures.
- Is it useful?
  - In controlled network environment applications should be configurable to use well known ports and traffic to unknown ports should be dropped.
  - Mostly useful in environment where users cannot be tightly controlled (ISP, university campus, ...).
  - Or to search tunneled traffic.

Matching Transport layer patterns

- Internet Traffic Identification using Machine Learning, Jeffrey Herman & al. IEEE Globecom 06.
  - Example: Using naïve bayes classifier.
    - Set of n variables describing communications (flows): Called features: F_0,...,F_n
    - Set of classes (e.g. HTTP, FTP, POP, P2P, ...): C
    - We want to compute P(C|F_1,...,F_n):
      \[ p(C = c | F_1 = f_1, ..., F_n = f_n) = \frac{p(F_1 = f_1, ..., F_n = f_n | C = c) p(C = c)}{p(F_1 = f_1, ..., F_n = f_n)} \]

- When receiving a flow characterized by F_1 = f_1, ..., F_n = f_n look for c maximizing:
  \[ p(F_1 = f_1, ..., F_n = f_n | C = c) = \prod_{i=1}^{n} p(F_i = f_i | C = c) \]

  - Naïve Bayes classifiers provide ~80% correct classification.
  - Using more complex classification techniques can yield 90-95% correct classification.
Matching Transport layer patterns

- Alternatives
  - Heuristics on port/addresses relations (BLINC, multilevel traffic classification in the dark, Karagiannis & al. SIGCOMM 05).

- Can we infer more than the protocol type?
  - Once you know the protocol you can apply similar techniques to obtain application level information.
    - Using bayesian networks.
    - For HTTP parameters matching.

Protocol Messages Parsing

- Protocols are specific forms of languages.
  - Formal constructs that are used to parse languages can be used to parse protocols.
  - State automaton, formal grammars ...
  - Automated techniques can be build parsers from formal descriptions.

Applying to HTTP parameters matching.

Application level Filter

- Overview

Normalization

- Usually written by hand.
- Main protocols:
  - HTTP:
    - % encoding (& double encoding).
    - Unicode encoding.
    - Directory traversal.
    - Upper-lower case.
    - Whitespace/Line feed
  - SMTP/DNS:
    - Unicode encoding.
    - Upper-lower case.
Normalization

- Main protocols:
  - Telnet/FTP/SSH
    - Control Codes (Backspace, Bell, CR, tabulation, …).
    - Commands (set of codes that are used to control the communication: termination, echo mode, window size, flow control, set environment variables, …).

Pattern Matching

- Several types of matching problems
  - String matching.
    - String = finite sequence of characters over finite alphabet.
    - Matching = search all occurrences of string over a text (longer string).

<table>
<thead>
<tr>
<th>RF</th>
<th>Src Address</th>
<th>Dst Address</th>
<th>Src Port</th>
<th>Dst Port</th>
<th>Proto</th>
<th>Method</th>
<th>URI</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>157.159.226.132</td>
<td>&gt;1024</td>
<td>80</td>
<td>HTTP</td>
<td>GET</td>
<td>Test.html</td>
<td>Deny</td>
<td></td>
</tr>
</tbody>
</table>

Pattern Matching

- Several types of matching problems
  - Multiple strings matching.
    - Multiple strings = Set of strings.
    - Matching = search all strings occurrences over a text reading the text once.

<table>
<thead>
<tr>
<th>RF</th>
<th>Src Address</th>
<th>Dst Address</th>
<th>Src Port</th>
<th>Dst Port</th>
<th>Proto</th>
<th>Method</th>
<th>URI</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>157.159.226.132</td>
<td>&gt;1024</td>
<td>80</td>
<td>HTTP</td>
<td>GET</td>
<td>Test.html</td>
<td>Deny</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>157.159.226.132</td>
<td>&gt;1024</td>
<td>80</td>
<td>HTTP</td>
<td>GET</td>
<td>Toast.html</td>
<td>Permit</td>
<td></td>
</tr>
</tbody>
</table>
Pattern Matching

- Several types of matching problems
  - Regular expressions matching.
    - Regular expression = concatenation, union, repetitions of strings or regular expressions.
    - Matching = search all regular expressions occurrences over a text reading the text once.

- Why not just using regular expressions for every search?

<table>
<thead>
<tr>
<th>R#</th>
<th>Src Address</th>
<th>Dst Address</th>
<th>Src Port</th>
<th>Dst Port</th>
<th>Proto</th>
<th>Method</th>
<th>URI</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>157.159.226.132</td>
<td>&gt;1024</td>
<td>80</td>
<td>HTTP</td>
<td>POST</td>
<td>*.htm</td>
<td>Deny</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td>157.159.226.132</td>
<td>&gt;1024</td>
<td>80</td>
<td>HTTP</td>
<td>GET</td>
<td>*.htm</td>
<td>Permit</td>
</tr>
</tbody>
</table>

Single String Matching

- Naïve search (prefix based).

  w

  A B D C A B A B Q A B D F C C D
  A B D F

  - When we don’t have a match we shift by 1

  w

  A B D C A B A B Q A B D F C C D
  A B D F

  - Until number of character matched = size of the window.

  w

  A B D C A B A B Q A B D F C C D
  A B D F

- Morris-Pratt (prefix based).

  w

  A B D C A B A B Q A B D F C C D
  A B D F

  - In this case we can shift by 4 since there is no part in w that matches a prefix of the pattern.

  A B D C A B A B A F B D F C C D
  A B A F

  - In this case we can only shift by 2 since a part of w matches the AB prefix of the pattern.

  A B D C A B A A F B D F C C D
  A B A F

  - We can start the search at B since A has already been recognized.
Single String Matching

- **Morris-Pratt (prefix based).**
  - In general how much can we shift?
  - The number of characters to align the prefix to the suffix found.
  - The shift only depends on the content of the pattern!

```
ABDABABQABDFCCD
ABDABF
```

```
Shift(i+1) = ((i + 1) – 1) – b = i - b
```

- **Knuth-Morris-Pratt.**
  - Avoid by noticing that D/Q can’t match A since the character tested prefix of the pattern.

---

Single String Matching

- **Morris-Pratt (prefix based).**
  - In general how much can we shift?
  - Pre-compute border b: the longest prefix v of pattern p that is also a suffix u during a search, u = v != p.

```
ABDABF
ABDAB
ABD
```

- **Boyer-Moore (suffix based).**
  - Full Good suffix shift.
  - If we know that a suffix of the pattern is repeated within the pattern.
  - We can shift the pattern to that position.
  - Since we have a mismatch between A and D the character before the searched suffix must be different from A.
  - If the pattern had been ABABA we could have shifted 5 characters.
Single String Matching

- Boyer-Moore (suffix based).

\[ w \]
\[ A B D B A B A B Q C A B A B A D \]
\[ A C A B A \]

- Partial Good suffix shift.
  - The full suffix is not repeated in the pattern but
  - A part of the recognized text suffix is also a prefix of the pattern
  - We want to align the prefix with this suffix.
  - Full good suffix prevails over partial good suffix.
  - There’s no need to align internal A since it is not preceded by B.

\[ w \]
\[ A B D B A B A B Q C A B A B A D \]
\[ A C A B A \]

- Full window Bad character shift.
  - The mismatched character (B) doesn’t occur in the rest of the pattern.
  - We can shift by the length of the window.
  - Any i position shift \((i < w)\) would generate a mismatch.

\[ w \]
\[ A B D B A B A B Q C A B A B A D \]
\[ A D A D A \]

- Combine the shifts.

\[ \text{Max}\left( \text{GS}(p) = \min \left( \text{GS}_{s}(p): \text{Good full suffix shift}, \text{GS}_{p}(p): \text{Good Partial suffix shift} \right) \right) = \left( \text{BF}_{s}(p): \text{Full window Bad character shift}, \text{BP}_{s}(p): \text{Partial window Bad character shift} \right) \right) \]

- Example: Pattern = GCAGAGAG, Alphabet = G,C,A,T

\[ \text{GSs} \]
\[ GCAGAGAG \]
\[ 388828481 \]

\[ \text{GSp} \]
\[ GCAGAGAG \]
\[ 77777777 \]

\[ \text{BS} \]
\[ GCAT \]
\[ 2618 \]

Non(G)AG appears is CAG
Non(G)AGAG appears is CAGAG
The only suffix that is also a prefix is G
**Single String Matching**

- Horspool (any order based).
  - Simplification of Boyer-Moore.
  - Only uses bad character shift.
  - Can work in any order (prefix or suffix based).
  - Works well with large alphabets/small patterns.
  - Reduces preprocessing and memory cost.

**Multiple String Matching**

- Aho-Corasick (prefix based).
  - Extension of KMP.
  - Builds automaton from set of searched patterns.
  - Root node, Intermediate node
  - Transitions.
  - Example: “toast.html|test.html|test.htm|toast.htm|oast”

**Multiple String Matching**

- Aho-Corasick (prefix based).
  - Terminal nodes: Node reached when a pattern is recognized.
  - Supply function.
    - State reached when reading the longest prefix that is also a suffix of the window.
    - Supply function is also terminal.
Multiple String Matching

- Aho-Corasick (prefix based).
  - Functions:
    - Transition function $T(node, character)$.
    - Supply function $S(node)$.
  - Searching occurrences:
    Current = root node.
    While there is a character $x$ to read
    While ($T(Current, x)$==0) && ($S(Current)$!=0)
      Current = $S(Current)$;
    If ($T(Current, x)$!=0)
      Current = $T(Current, x)$;
    Else
      Current = root node;
    If Current is terminal
      mark occurrence;

Some other important algorithms
- Horspool
  - Efficiency reduces as the number of strings increases since the likelihood of bad character shift decreases exponentially with number of strings.
- Wu-Manber
  - Improves over Horspool.
  - Uses character blocks in order to increase bad character shift efficiency.
  - Uses hash table in order to avoid storing $|\Sigma|^B$ shifting values. ($B$: number of characters per block, $\Sigma$: Alphabet).

Regular expression Matching

- Definition.
  - A regular expression is a string on the set of symbols $\Sigma \{ \epsilon, \ldots, \ast, (, ) \}$ which is recursively defined as the empty character $\epsilon$, a character $a \in \Sigma$ and $(RE_0 \ast (RE_1))$, $(RE_0 | RE_1)$, $(RE_0 \ast RE_1)$ and $(RE_1 \ast RE_0)$ where $RE_0$ and $RE_1$ are regular expressions.
  - Example: $((A.B)\ast((A.C)\ast(A.D))) = ((AB)\ast((AC)\ast(AD)))$
  - Usually $(RE_0 \ast RE_1)$ is simplified to $RE_0RE_1$
Regular expression Matching

- Definition.
  - The language represented by a regular expression RE is a set of strings over $\Sigma$ which is recursively defined as follows:
    - If RE is $\epsilon$ then $L(RE) = \{\epsilon\}$, the empty string.
    - If RE is $a \in \Sigma$ then $L(RE) = \{a\}$, a one character string.
    - If RE is of the form $(RE_0)$ then $L(RE) = L(RE_0)$.
    - If RE is of the form $(RE_0 | RE_1)$ then $L(RE) = L(RE_0) \cup L(RE_1)$, $L(RE)$ where $X \cup Y$ is the set of strings $w=x.y$ with $x \in X$ and $y \in Y$. $L(RE)$ is the language formed by the concatenation of strings generated by languages $L(RE_0)$ and $L(RE_1)$.
    - If RE is of the form $(RE_0)*$ then $L(RE) = L(RE_0)* = \cup_{i \geq 0} L(RE_0)^i$ where $L(RE_0)^0 = \{\epsilon\}$ and $L(RE_0)^i = L(RE_0) . L(RE_0)^{i-1}$.

Regular expression Matching

- Non-deterministic Finite state Automaton.
  - Thompson automaton

Regular expression Matching

- Regular expression matching process.
  - Parsing
  - NFA construction
  - Determinization

- Parse tree generation.
  - Using lexical & syntactic analyzers.
  - Building analyzer by hand.

Non-determinism:
- $\epsilon$ transitions or
- Several transitions from a state with the same label
Regular expression Matching

- Non-deterministic Finite state Automaton.
  - Searching with Thompson automaton.
  - Several states can be simultaneously active.
  - The automaton only recognizes the language described by the regexp.
  - We need transitions allowing us to be in initial state when a pattern is not recognized.

<table>
<thead>
<tr>
<th>Letters Red</th>
<th>Active States</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>I</td>
</tr>
<tr>
<td>/</td>
<td>1, 5, 7</td>
</tr>
<tr>
<td>A</td>
<td>1, 5, 7</td>
</tr>
<tr>
<td>AB</td>
<td>2</td>
</tr>
<tr>
<td>ABA</td>
<td>1, 5, 7</td>
</tr>
<tr>
<td>ABAB</td>
<td>2</td>
</tr>
<tr>
<td>ABABA</td>
<td>1, 5, 7</td>
</tr>
<tr>
<td>ABABAA</td>
<td>2</td>
</tr>
<tr>
<td>ABABABA</td>
<td>1</td>
</tr>
<tr>
<td>ABABABA</td>
<td>1, 5, 7</td>
</tr>
<tr>
<td>ABABABA</td>
<td>2</td>
</tr>
<tr>
<td>ABABAAB</td>
<td>1, 5, 7</td>
</tr>
<tr>
<td>ABABAABC</td>
<td>F, I</td>
</tr>
</tbody>
</table>

Text to read: ABABAABC

- Deterministic Finite state Automaton.
  - \(\varepsilon\) closure definition:
    - The \(\varepsilon\) closure of a state \(s\) in an NFA \(E(s)\) is the set of states of the NFA that can be reached from \(s\) with \(\varepsilon\) transitions.
  - \(\varepsilon\) closure example:

  - \(E(I) = \{I, 1, 4, 5, 7\}, E(3) = \{3, 1, 4, 5, 7\}, E(4) = \{4, 5, 7\}, E(6) = \{6, F\}, E(8) = \{8, F\}\)
  - \(E(1) = \{1\}, E(2) = \{2\}, E(5) = \{5\}, E(7) = \{7\}\)

Text to read: ABABAABC

Regular expression Matching

- Deterministic Finite state Automaton.
  - \(E(I)\) is the initial state of the DFA.
    
    ```
    Build_state(S)
    For every \(a \in \Sigma\),
    For every NFA states \(s \in S\),
    If there is a transition \(a\) between \(s\) and \(s'\), \(s'\) \(\in E(y)\) then
    Add \(E(y)\) to DFA state \(T\);
    Build_transition(S, T, a);
    If \((T\) has not been already treated) then
    Build_state(T);
    ```
  - A state is final if an NFA final state is included in it.

Text to read: ABABAABC
**Regular expression Matching**

- Deterministic Finite state Automaton.
  - Searching
    - We have the same problem we met using the NFA.
    - The DFA should have been built from the modified NFA we saw earlier.

Text to read: ABABAABC

**String Matching Complexities**

- **Complexities**

<table>
<thead>
<tr>
<th>Method</th>
<th>Best Case</th>
<th>Worst Case</th>
<th>Average Case</th>
<th>Precomputation</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>O(n)</td>
<td>O(n.m)</td>
<td>O(n.m)</td>
<td>O(0)</td>
<td>O(0)</td>
</tr>
<tr>
<td>MP/KMP</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(m)</td>
<td>O(m)</td>
</tr>
<tr>
<td>Boyer-Moore</td>
<td>O(n/m)</td>
<td>O(n.m)</td>
<td>O(n)</td>
<td>O(m)</td>
<td>O(2^</td>
</tr>
<tr>
<td>Horspool</td>
<td>O(n)</td>
<td>O(n+nocc)</td>
<td>O(n+nocc)</td>
<td>O(m)</td>
<td>O(2^</td>
</tr>
<tr>
<td>Wu Manber</td>
<td>O(n/m,B)</td>
<td>O(n.m)</td>
<td>O(n)</td>
<td>O(2^n)</td>
<td>O(2^n)</td>
</tr>
<tr>
<td>NFA</td>
<td>O(n.m)</td>
<td>O(n.m)</td>
<td>O(n.m)</td>
<td>O(m)</td>
<td>O(m)</td>
</tr>
<tr>
<td>DFA</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(2^n)</td>
<td>O(2^n)</td>
</tr>
</tbody>
</table>

- **Bit level parallelism:**
  - Takes advantage of multi-characters comparison on modern processors (16-128 bits).
  - Divides search time by memory bus width.
  - Transforms search in bitlevel operations.

**Application level Filter**

- **Overview**

  - Filtering Policy
  - State Checking
  - State Lookup
  - Pattern Matching
  - Normalization
  - Protocol Analysis
  - Application-Level Information

  **Incoming Packets**
  - Internal Interface
  - Filter
  - External Interface
  - Authorized Packets
**Application level Filter**

- Overview

  - Filtering Policy
  - State Checking
  - Pattern Matching
  - State Lookup
  - Normalization
  - Protocol Analysis
  - Application-Level Information

**State Checking**

- Several aspects (complementary).
  - Checking that protocol messages sequencing is compliant with protocol specification.
  - Checking that message does not contain known attacks.
    - Similar to knowledge based IDS.
  - Checking that protocol usage is compliant with legitimate users behavior.
    - Similar to behavior based IDS.

**Authentication**

- Motivations
  - Improve connection between users and addresses.

1. User authenticates to authentication server.
2. Authentication server relates IP address and user name to generate filtering rules according to users rights. Rules are sent to filtering module.
3. User communicates with external device.
Examples

- Ipfilter
  - Works on most Unixes.
  - Public source licence (Not BSD).

- Functionnalities
  - Stateless Packet filtering.
  - True stateful packet filtering.
  - NAT (NAT, PAT).
  - Transparent proxying.

- Organization
  - Filtering rules stored in a configuration file.
  - Software uses configuration file to configure filtering module.

IPFilter

- Architecture

  * Fragment cache check.
    - Ipfilter does not deal with packets when frag#0 does not contain transport information.
    - These packets are not considered legitimate.
    - Fragment #0 contains IP+Transport Information.
    - A state is kept in the state table iff:
      - Fragment #0 matches classification “permit” rule.
      - Rule indicates that fragment state should be kept.
    - Fragment #n n>0 are
      - Matched against fragment states.
        - If a match is found:
          - Used to update fragment states table.
          - Either forwarded to destination bypassing other checks or dropped.
        - If no match is found:
          - Dropped.
    - All forwarded fragments are sent as is (no reassembly).
IPFilter

• Architecture

Fragment Check
Packet Route Check
Port Check
Firewall Check
IP Accounting
Network Address Translation
pass only

OUT

Conclusion

• Packet filters efficiency depends on many parameters.
  – Location.
  – Filtering process.
  – Administration.
  – Interaction with other security components.

• Packet filters challenges.
  – Dynamic applications (no fixed ports).
  – End to end encryption.
  – Performance (Internal bus, states management, dynamic policies).
  – New protocols (STCP, …)

Thank You!