Problem Statement

- There is a demand for fine-grained accounting
- two drafts already exist
- the solution is obvious
How standards proliferate:
(see: A/C chargers, character encodings, instant messaging, etc)

**Situation:**
There are 14 competing standards.

14?! Ridiculous!
We need to develop one universal standard that covers everyone's use cases.

Yeah!

**Soon:**

**Situation:**
There are 15 competing standards.
The real problem

• Why is RADEXT defining what information should be used for accounting traffic flows?

• RADIUS can transport data.

• Defining “this is a traffic flow” is really outside of the scope of RADEXT
The Solution

- Benoit noted IPFIX
- [http://www.iana.org/assignments/ipfix/ipfix.xml](http://www.iana.org/assignments/ipfix/ipfix.xml)

<table>
<thead>
<tr>
<th>ElementID</th>
<th>Name</th>
<th>Data Type</th>
<th>Data Type Semantics</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>octetDeltaCount</td>
<td>unsigned64</td>
<td>deltaCounter</td>
<td>current</td>
</tr>
<tr>
<td>2</td>
<td>packetDeltaCount</td>
<td>unsigned64</td>
<td>deltaCounter</td>
<td>current</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
XML Registry? Perl!

```perl
#!/usr/bin/env perl
use XML::Simple;
use Data::Dumper;

$prefix = "TBD";  # IANA - to be updated
print "ATTRIBUTE IPFIX-Container $prefix long-extended\n";

$map = {
    'unsigned8'   => 'integer',
    'unsigned16'  => 'integer',
    'unsigned32'  => 'integer',
    'unsigned64'  => 'integer64',
    'ipv4Address' => 'ipaddr',
    'ipv6Address' => 'ipv6addr',
    'string'      => 'text',
    'octetArray'  => 'string',
    'dateTimeSeconds' => 'date',
    'dateTimeMilliseconds' => 'integer64',
    'dateTimeMicroseconds' => 'integer64',
    'dateTimeNanoseconds' => 'integer64',
    'macAddress'  => 'string',
    'boolean'     => 'integer',
};

$xml = new XML::Simple;
$data = $xml->XMLin("ipfix.xml");
$elements = $data->{registry}->{ipfix-information-elements}->{record};

foreach $record (@$elements) {
    next if $record->{unassigned};
    next if $record->{reserved};
    $name = $record->{name};
    $name =~ s/\s+//g;
    $upper = 1 + (($record->{elementId} & ~0xff) >> 8);
    $lower = $record->{elementId} & 0xff;
    $oid = "$prefix.$upper.$lower";
    $type = $record->{dataType};
    if (defined $map->{$type}) {
        print "ATTRIBUTE IPFix-$name $oid ", $type, "\n";
    } else {
        print "ATTRIBUTE IPFix-$name $oid ", $map->{$type}, "\n";
    }
}
```
Output

- ATTRIBUTE IPFIX-Container TBD long-extended
- # ATTRIBUTE IPFix-Reserved TBD.1.0
- ATTRIBUTE IPFix-octetDeltaCount TBD.1.1 integer64
- ATTRIBUTE IPFix-packetDeltaCount TBD.1.2 integer64
- ATTRIBUTE IPFix-protocolIdentifier TBD.1.4 integer
- ATTRIBUTE IPFix-ipClassOfService TBD.1.5 integer
- ATTRIBUTE IPFix-tcpControlBits TBD.1.6 integer
- ATTRIBUTE IPFix-sourceTransportPort TBD.1.7 integer
- ATTRIBUTE IPFix-sourceIPv4Address TBD.1.8 ipaddr
- ATTRIBUTE IPFix-sourceIPv4PrefixLength TBD.1.9 integer
- ...

...
The benefit

- We leverage the existing IPFIX registry
- Can do flow-based accounting for any flow
- MPLS, TCP, UDP, deltas, absolute counters, etc.
- No need to re-invent the wheel
The drawback

- IPFIX has 16-bit IDs
- OK, a bit of work and we can hack them into RADIUS
- Some other IPFIX things aren’t relevant either
- Security attributes, etc.
- probably 99% of the IPFIX attributes are relevant and useful in RADIUS
The Conclusion

• We should specify transport, not content
• Once we publish a draft, the entire accounting problem will go away
• Never to return
Discussion?