DPI traffic classification

Niccolo' Cascarano

niccolo.cascarano@polito.it
DPI: classification or verification?

• “Verification” vs “classification”
  – Classification was in the old days, when application were detected by “classifying” the packet based on a field
  – Now, we need to verify that the packet keeps to some well defined rules

  ➔ Classification = Verification

• Protocol verification, or application verification?
  – Skype can use the standard HTTP protocol to get data
  – Is that traffic “Skype” or “HTTP”?
Protocol verification

- **Different degrees of verification**
  - Protocol is X and not Y
  - Protocol is X, but not well formed

- **Signature-based verification**
  - "Does the message contain some well-known pattern?"
  - Signature + other "simple" checks (e.g., packet length, ...)
  - Can be seen as a *lightweight* syntactical verification, particularly if coupled with TCP and IP normalization

- **Syntactical verification**
  - "Is the message well-formed?"

- **Protocol Conformance verification**
  - "Is the protocol state machine coherent?"

- **Semantic verification**
  - "Is the HTTP packet transporting an image, as it was declared in the header?"
Some notes about protocol verification

• Protocol verification, or application detection?
  – Skype can use the standard HTTP protocol to get data
  – Is that traffic “Skype” or “HTTP”?

  ➔ Even the Protocol Conformance Verification cannot get an answer to this problem

• Let’s call this problem “application detection”
  – One step further (is it orthogonal to protocol verification?)
Protocol verification and pre-classification

• **Verification**
  - “Is the protocol what was it supposed to be?”

• **Pre-classification**
  - A smart method to pick the most likely protocol first
    - *Usually reduces the number of protocol to verify*
  - Most-common approaches
    - *Port-based*
    - *Signature-based*
  - Used by most technologies (also not signature-based)
Payload-based traffic classification

• **Tools for better Packet Classification**
  - Session Tracking (see next slide)
  - IP and TCP reassembler
    • *Requires lot of resources, but there is nothing more to say*
  - Protocol state machine
    • *Better if we have a language for describing the protocol state machine*

• **Other issues that need to be addressed**
  - Application-negotiated sessions
Application-negotiated sessions

- Need to parse application-layer protocol for “child” session ID
  - FTP, SIP
- Incomplete entries
  - E.g. FTP passive
- Needs some “protocol parsing” capabilities
  - Often done also through regex
**Session tracking**

- **“TCP session table”**
  - Keeps the correlation among packets belonging to the same session

- **Packet-based technologies**
  - Use ST to correlate packets (e.g. Sequence Numbers in RTP)
  - Use ST to keep the state of the verification
    - *The signature may be present only on the first packet*
    - *Next packets must not be verified*

- **Stream-based technologies**
  - Use ST to point to the proper instance of the TCP reassembler

- **Needs to address the “incomplete entry” problem**
  - “masked” entries
And what about application detection?

- **This is something orthogonal compared to protocol verification**
- **Usually done through some signature-based analysis**
  - E.g., if the “user-agent” is “emule”, probably this is an edonkey download through HTTP
  - At this time, no differences between different protocol verification technologies
Some existing technologies

- **SML, BinPac**
  - Stream-based
  - Cannot be used packet-by-packet

- **NetPDL**
  - Currently packet-based, but it strongly depends on the implementation
  - It has also primitives for packet-based analysis (e.g. `<missing-data>` element)

<table>
<thead>
<tr>
<th></th>
<th>Signature-based preclass.</th>
<th>Signature-based</th>
<th>Signature-based + TCP/IP normalization</th>
<th>Syntactical verification</th>
<th>Behavioural verification</th>
<th>Application verification</th>
<th>Semantic verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinPac</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SML</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>NetPDL</td>
<td>Yes</td>
<td>Yes</td>
<td>Planned</td>
<td>Possible (with extensions)</td>
<td>Possible</td>
<td>Partially implemented</td>
<td>No</td>
</tr>
</tbody>
</table>
NetPDL and TCP normalization

- Ethernet frame
- HTTP virtual packet
- HTTP headers
- HTTP payload
- NetPDL processing engine
NetPDL: simple example

```xml
<protocol name="Ethernet" longname="Ethernet 802.3">
  <format>
    <fields>
      <field type="fixed" name="dst" longname="MAC Dest." size="6"/>
      <field type="fixed" name="src" longname="MAC Source" size="6"/>
      <field type="fixed" name="type" longname="Ethertype" size="2"/>
    </fields>
  </format>

  <encapsulation>
    <switch expr="buf2int(type)">
      <case value="0x0800"> <nextproto name="#IP"/> </case>
      <case value="0x0806"> <nextproto name="#ARP"/> </case>
    </switch>
  </encapsulation>

</protocol>
```

Header format

Protocol encapsulation
NetPDL and TCP encapsulation

Dynamic entries
<if expr="checklookuptable($tcpsessiontable, ...)">
  <if-true>
    <nextproto proto="$tcpsession.nextproto"/>
  </if-true>
</if-

“Well-known” entries
<switch expr="sourceport">
  <case value="21"><nextproto-candidate proto="#ftp"/></case>
  <case value="80"><nextproto-candidate proto="#http"/></case>
</switch>
<switch expr="destport">
  <case value="21"><nextproto-candidate proto="#ftp"/></case>
  <case value="80"><nextproto-candidate proto="#http"/></case>
</switch>

"Try and see" entries
<nextproto-candidate proto="#http"/>
<nextproto-candidate proto="#skype"/>
Evaluating the results

Parameters

Coverage
- # of supported protocol

Accuracy
- Quality of the classification for protocol X

False positives
- “this proto is X”, while it is not

False negatives
- “this protocol is not X”, while it is

Unclassified
- “this proto is unknown”, while it is X

Wrong classification
- “this proto is Y”, while it is X

It has to be done “per protocol”

Only for protocol in our “coverage” domain
Coverage

- **Current status**
  - A large number of supported protocols
    - 122 protocols detected
    - Several P2P applications
  - The quality of the classification may vary
    - In fact, this is a very huge amount of work

- **Comments**
  - We can say that ALL IETF protocols are (or can be) supported
  - All the protocol that are “well-behaving” (also P2P with dynamic ports, ...) can be supported
  - We cannot support encrypted protocols
    - But all the payload-based technologies have this limit
Comments about accuracy

• **The accuracy is very high**

  So far, it is influenced only by the quality of the “signature”
  The NetPDL language provides all the mechanisms we need

• **Almost all the errors are usually related to missing signatures**
  – It’s a matter of signatures, even if a protocol wants to screw you up

• **False Positives depend on the protocol (e.g. much likely for HTTP)**

• **Limits related to the PacketBased-PerFlowState (PBFS)**
  – Signature split across two packets
  – Missing protocol state machine
  – Do not seem to be relevant if no 100% accuracy is required
“Fully stateful” classification: BRO

- **Signature used for detecting protocols on unknown ports**
  - Unfortunately, only a few protocols have this feature
  - http, smtp, ftp, ssh, irc

- **Protocol conformance classification**
  - Through BinPac dissectors
  - dns, dce_rpc, http, ncp, rpc, smb

- **For our tests, the only suitable protocol is HTTP**
  - It is detected also on non-default ports (thanks to the signature)
  - It is parsed through a BinPac dissector (behavioral classification)
Some comments

• “fully stateful” classification is
  – More precise
    • ...as expected, if everything is correct
    • Packets with “unidirectional signatures” are, in fact, not valid HTTP packets
  – Less robust, e.g., to packet losses
    • Losses in the network, low CPU (see Telecom Italia)
  – Can be used only on the access slide
    • Asymmetric routing problem

• BRO
  – Does not implement “application detection”
  – So, some sessions (which are valid HTTP sessions indeed) are not recognized as “skype traffic”
Conclusions

• **PacketBased-PerFlowState vs MessageBased-PerProtocolState**

• **From QoS perspective**
  - MBPS has more, but...
    - *Syntactical verification: not important for QoS*
    - *Stream based (splitted signatures): we can do without it (see current accuracy results)*
    - *Behavioural verification: idem*
  - PBPS has everything you need, with enough (very high) accuracy
  - Please note that NetPDL is PBFS, but it can become MBFS

• **From the security perspective**
  - The story is very different
Some links

• **NetBee Library**
  - http://www.nbee.org

• **Protocol database and NetPDL latest specs**
  - http://www.nbee.org/netpdl/