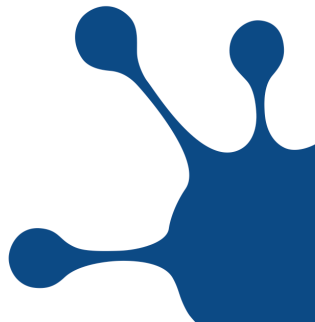
DSC on top of PacketQ

Johan Ihrén
Netnod

October 14, 2012

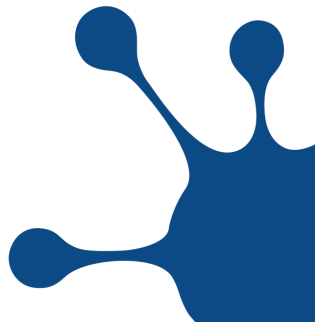


DSC

- Why DSC?
  - De-facto standard, especially among TLDs
- Why not DSC?
  - Design choices for DSC may not be right for everyone
  - Integration with other systems and services
  - Performance
- However, as long as we keep the DSC data format in the interface to the customer we're fine.
- I.e. we want to leverage from the wide spread usage of DSC by enabling customers to keep their presentation tools while allowing us to run a completely different collection infrastructure

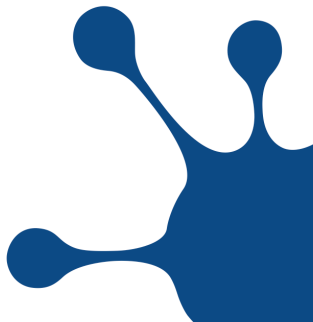
The DNS Data Collection Problem

- Our problem
  - There are many sites
  - There are many packets per second
  - There is lots of disk churn on the collector
  - There is finite bandwidth back to the office
- Our conclusion was that we must optimize for the choke points, like disk bandwidth on the collector, network bandwidth from the site, etc
- The DSC design, at least at the time, was not geared towards those needs and hence we had to roll our own



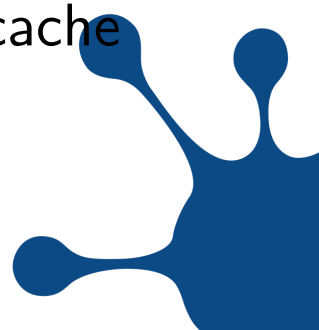
PacketQ

- **PacketQ** is a piece of software that basically applies one (or more) SQL queries to the contents of a **pcap** file and return the result as a JSON structure
  - **PacketQ** is originally developed by .SE and is open source
- We use **PacketQ** to enable us to do data mining in captured data without having to first process it into another format
  - Standard DSC process **pcap** data into XML
  - **DNS2DB** (which we're currently do use, but are phasing out) process **pcap** data into SQLite format
  - Such processing is costly



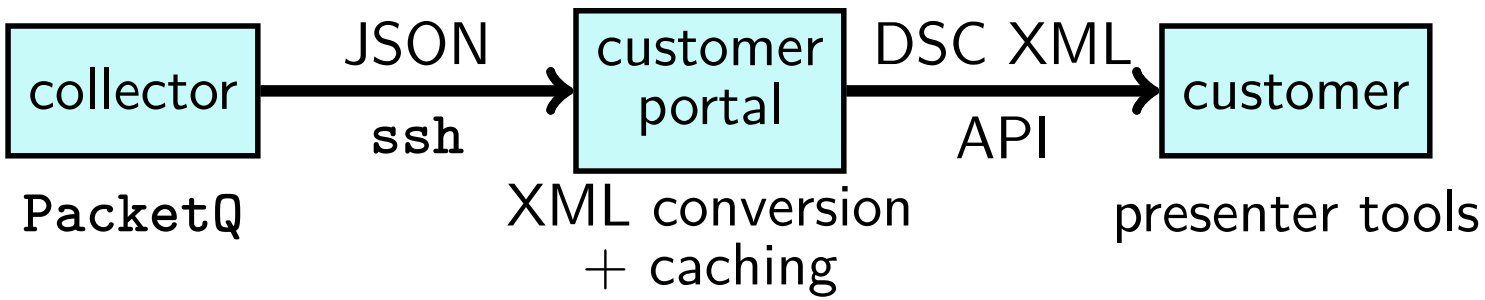
Comparision between DSC Collection and What We Do

- DSC proactively process all the `pcap` data on the collector. We don't touch the `pcap` data until there's a request for it
- DSC converts `pcap` data into XML and stores this on local disk before shipping it towards the presenter. We query the `pcap` data directly (via the SQL interface provided by `PacketQ`) and return the result without any intermediate storage on collector disks
- DSC uses a **push** mechanism for data retrieval where the collectors send the XML files back to the presenter. We use a **pull** design where we query for what is requested (and cache it centrally)



Comparision, cont'd

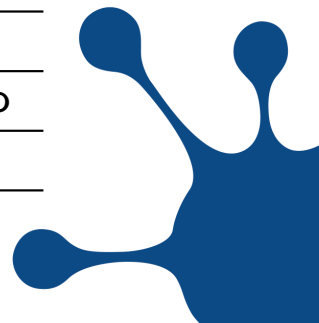
- The internal data representation is JSON rather than XML
- As the interface to the data is always via our "customer portal" (which exists both as a web interface and as an API) we synthesize DSC XML format in the customer portal
  - We also cache all data in case it will be queried for again



Implementation Issues

- Mostly, we had no problems implementing support for the DSC datasets that we presently support
- The primary exception is the implementation of "`query_classification`", which is used to break down "**bogus**" queries into different sub-types:

sub-type	**description**
`non-auth-tld`	not on the list of IANA-approved TLDs
`root-servers.net`	a query for a root server IP address
`localhost`	a query for the localhost IP address
`a-for-root`	an A query for the DNS root (.)
`a-for-a`	an A query for an IPv4 address
`rfc1918-ptr`	a PTR query for an RFC 1918 address
`funny-class`	a query with an unknown/undefined query class
`funny-qtype`	a query with an unknown/undefined query type
`src-port-zero`	when the UDP message's source port equals zero
`malformed`	un-parseable packet



Implementation Issues: Bogus Queries

- There is a compiled-in and out-of-date list of "IANA-approved TLDs". F.i. queries for "**`whatever.eu.`**" count as bogus in spite of **`.EU`** being a customer of ours for a long time.
- The DSC implementation of "**`non-auth-tld`**" doesn't look att the **`qclass`** and hence valid queries for "**`id.server. CH`**" and similar are counted.
- Also other bugs in the latest version of DSC code that we've looked at (March 2012) and we'll feed that back to the authors

The real issue here is not bugs as such, but rather that we have to be "**bug compatible**". I.e. we cannot fix this except in sync with the official DSC code. Hence, I'd be happy to discuss redefining classification of "**`bogus`**" with others...

Future

- A primary reason for using our own collection infrastructure is to be able to do other things than DSC is designed for
  - In particular, we want to keep more data than DSC does
  - The services we currently provide via DNS2DB ("most queried name", "most active resolver", etc) are being re-implemented on top of `PacketQ`
- Missing DSC "datasets" are implemented (as `PacketQ` SQL queries) as we get requests for them
- We have not yet done much work on the presenter tools (as customers mostly use their own)
- Future Tools
  - While services like "most queried name", etc, are already being implemented the possibilty of more generalized data mining remain an interesting possibility