Toward Efficient Querying of Compressed Network Payloads

By Teryl Taylor and Fabian Monrose
University of North Carolina at Chapel Hill
Scott E. Coull and John McHugh
RedJack
Motivation

Please Log onto http://cs.unnc.edu and change your password
Problem: How does an analyst perform efficient forensic analysis using all data available to him/her? Time is of the essence.
Goal: Build an interactive query system for network traffic which supports payload-based queries.

Challenges:

- Large volume of network traffic: billions of records (TBs of data).
- Network data has high dimensionality – i.e. lots of attributes.
- Payloads are heterogeneous.
Existing Approaches

- Scan and Filter approaches (e.g. TCPDump).
  - Write optimized 😊.
  - Must scan each record in the dataset 😞.

- Parallel Scan and Filter approaches (e.g. MapReduce).
  - Requires significant computational resource 😞.
  - Google: Data organization is important for speed 😞.

- Relational Databases (row-wise) (e.g. Postgres).
  - Rigid structure 😞.
  - Slow / bloated index generation 😞.
Approach

- Build a low I/O bandwidth (read-optimized) storage and query framework.

- Utilizing the following principals:
  - Reduce data footprint: aggregation and compression.
  - Only touch data relevant to the query: indexing.
  - Reduce disk I/O: horizontal and vertical partitioning.
  - Support arbitrary application protocols: flexible schemas.
SELECT SourceIP, Dns.Query.Type

Locate Data Partitions

Locate Records

Return Records
Approach

Reduce Data Footprint....
### Record Aggregation on Flows

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dest IP</th>
<th>Src Port</th>
<th>Dst Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.10.4.2</td>
<td>13.21.7.6</td>
<td>13</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>96.21.7.5</td>
<td>43.89.2.1</td>
<td>53</td>
<td>53</td>
<td>UDP</td>
</tr>
</tbody>
</table>

- Low bandwidth I/O 😊.
- Special read-only bitmap indices 😊.
Payload objects support flexible schema.
- Attributes are extracted using deep packet inspection.
- Each object has a template.
- Template instantiated and filled with attributes.
Joining Flows and Payloads

Column Oriented Flow + Serialized Payload Objects

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dest IP</th>
<th>Src Port</th>
<th>Dest Port</th>
<th>Protocol</th>
<th>Payload Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.10.4.2</td>
<td>13.21.7.6</td>
<td>13</td>
<td>80</td>
<td>TCP</td>
<td>HTTP</td>
</tr>
<tr>
<td>96.21.7.5</td>
<td>43.89.2.1</td>
<td>53</td>
<td>53</td>
<td>UDP</td>
<td>DNS</td>
</tr>
</tbody>
</table>

Approach: Reduce Data Footprint

Querying of Compressed Payloads
Payload Compression

- Two forms of compression
  - Record (object) level dictionary compression.
  - Block-level compression using LZO.

<table>
<thead>
<tr>
<th>DNS Object 1</th>
<th>Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query.Domain</td>
<td>1 <a href="http://www.google.com">www.google.com</a></td>
</tr>
<tr>
<td>Answer.Response</td>
<td>2 10.51.13.45</td>
</tr>
<tr>
<td>Answer.Type</td>
<td>3 A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DNS Object 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query.Domain</td>
</tr>
<tr>
<td>Answer.Response</td>
</tr>
<tr>
<td>Answer.Type</td>
</tr>
</tbody>
</table>
Approach

Reduce Disk I/O....
Partitioning the Data

- Records grouped (partitioned) into configurable sizes.
- Records are added to partition as they expire.
- Partitions are independent of each other.

- Advantages:
  - Allows us to access only relevant partitions.
  - Enables data distribution to support query parallelization.
Select Matching Partitions with Root Indices

Data Fields

Flow Fields
- DestIP
- DestPort
- StartTime
- PktCount
- Protocol

Payload Fields
- SourceIP
- SourcePort
- Duration
- ByteCount
- TcpFlags

Flow Fields
- Http.Host
- Http.Method
- Http.UserAgent
- Http.Uri
- Dns.Answer.Type
- Dns.Answer.Domain
- Dns.Query.Type
- Dns.Query.Domain

Payload Fields

Indexing Data Structures

- Root Bitmap Index
  - Partition 1 Bitmap Index
    - Partition 1 Record Store
  - Partition 2 Bitmap Index
    - Partition 2 Record Store
  - Partition P Bitmap Index
    - Partition P Record Store

Reduce Disk I/O

Querying of Compressed Payloads
Approach

Only Load Relevant Records....
Partition Indices

- Indices important to avoid scan and filter.
- Each flow-based attribute has a bitmap index.
### Payload Indices

#### DNS Payload Object from Template
- **Query.Domain**: krb1.unc.edu
- **Query.Type**: A
- **Answer.Domain**: krb1.unc.edu
- **Answer.Type**: CNAME
- **NameServer**: isis.unc.edu

#### HTTP Payload Object from Template
- **Http.Method**: GET
- **Http.Uri**: /
- **Http.Host**: www.af.mil
- **Http.UserAgent**: Mozilla/5.0
- **Http.Version**: HTTP/1.0
- **Http.Language**: en
- **Http.Length**: 1107

---

#### Term | Record IDs
--- | ---
**Dns.Query.Domain** | |
krb1.unc.edu | ![Bitmap Indexes](1)
www.unc.edu | ![Bitmap Indexes](1)

**Http.Host** | |
www.facebook.com | ![Bitmap Indexes](1)
www.af.mil | ![Bitmap Indexes](1)

**Http.UserAgent** | |
Mozilla/5.0 | ![Bitmap Indexes](1)
Mozilla/2.0 | ![Bitmap Indexes](1)

---

**Only Load Relevant Records**

**Querying of Compressed Payloads**
SELECT SourceIP, Dns.Query.Type
WHERE
DestPort = 53 AND
Evaluation on Campus DNS Data and DNS/HTTP Data

- **Dataset 1: Campus DNS – 5 day**
  - Number of Connections: 325 million
  - Original Raw Trace: 122 GB
  - Data Store (uncompressed): 155 GB
  - Data Store (dictionary compressed): 83 GB
  - Data Store (dictionary + LZO compressed): 75 GB

- **Dataset 2: Campus DNS/HTTP – 2.5 hours**
  - Number of Connections: 11.1 million
  - Original Raw Trace: 400 GB
  - Data Store (dictionary + LZO compressed): 12 GB

*Dual Intel Xeon 2.27 GHZ, 12 GBs RAM, single 2TB 7200 RPM local SATA Drive*
Evaluation Queries

- **Heavy Hitter** Queries: return the majority of records.

- **Partition Intensive** Queries: return a few records from every partition.

- **Needle in a Haystack** Queries: return a few records.

# Flow-based Query Comparison

<table>
<thead>
<tr>
<th>Flow-based Queries</th>
<th>Heavy Hitters</th>
<th>Partition Intensive</th>
<th>Needle Haystack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postgres</td>
<td>18.1m</td>
<td>9.5m</td>
<td>2.0s</td>
</tr>
<tr>
<td>SiLK</td>
<td>1.8m</td>
<td>1.8m</td>
<td>1.8m</td>
</tr>
<tr>
<td>Our approach</td>
<td>2.5s</td>
<td>30.5s</td>
<td>0.04s</td>
</tr>
</tbody>
</table>

e.g., SELECT source_IP, destination_IP from all_traffic where protocol=UDP

* DNS dataset
Payload-based Query Comparisons with Relational Joins

<table>
<thead>
<tr>
<th>Payload-based Queries (joins)</th>
<th>Heavy Hitters</th>
<th>Partition Intensive</th>
<th>Needle Haystack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postgres</td>
<td>&gt; 2h</td>
<td>&gt; 2h</td>
<td>3s</td>
</tr>
<tr>
<td>Our approach</td>
<td>30m</td>
<td>2.5m</td>
<td>0.1s</td>
</tr>
</tbody>
</table>

e.g., SELECT `dns.query.type`, `dns.ns.ttl` FROM `all_traffic` WHERE `dns.queryDomain=www.facebook.com` AND `dns.ns.domain=ns.facebook.com`.

* DNS dataset
### Single Table Payload-based Query Comparisons

<table>
<thead>
<tr>
<th>Payload-based Queries (no-joins)</th>
<th>Heavy Hitters</th>
<th>Partition Intensive</th>
<th>Needle Haystack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postgres</td>
<td>7.6m</td>
<td>9.7m</td>
<td>1.6s</td>
</tr>
<tr>
<td>Our approach</td>
<td>9.7m</td>
<td>3.3m</td>
<td>0.1s</td>
</tr>
</tbody>
</table>

E.g., SELECT `dns.queryType` FROM `all_traffic` WHERE `dns.queryDomain=www.google.com` OR `dns.queryDomain=www.facebook.com`

* DNS dataset
Impact of Payload Compression

>3x Speed Gain

3x speedup Partition Intensive over Heavy Hitters
Impact of Payload Compression

LZO block must be decompressed before query
Summary

- Framework has been used recently to:
  - Finding DGA domains – including Cridex malware.
  - Blacklisted domains.
  - Fast-Flux analysis.
  - Finding correlations between HTTP and DNS requests.
- Sub minute flow query times.
- Sub second needle in a haystack queries.
- Partition Intensive queries on the order of a few minutes.
- Slow linear growth over time.
Conclusions

- Designed a low bandwidth, random access query framework for network traffic.
- Improved query performance over existing solutions.

Take Aways:
- Record-level compression provides significant data reduction and improved query performance.
- Limit object parsing for performance
- Data organization important for data reduction,
- Multi-level indexing improves query performance through I/O reduction.