Mothra: A Large-Scale Data Processing Platform for Network Security Analysis

Tony Cebzanov

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Agenda

• Introduction
• Architecture and Design
• Demonstration
• Future Work
Mothra: A Large-Scale Data Processing Platform for Network Security Analysis

Introduction
In the beginning... there was Netflow

Netflow was designed to retain the most important attributes of network conversations between TCP/IP endpoints on large networks without having to collect, store, and analyze all of the network's packet-level data

• For many years, this has been the most effective way to perform security analysis on large networks

• Over time, demand has increased for a platform that can support analytical workflows that make use of attributes beyond the transport layer

Modern flow collectors can export payload attributes at wire speed

• The challenge is scalable storage and analysis

• The current generation of distributed data processing platforms provides tools to address this challenge
Project Goals

The Mothra security analysis platform enables scalable analytical workflows that extend beyond the limitations of conventional flow records.

With the Mothra project, we aim to:

- facilitate bulk storage and analysis of cybersecurity data with high levels of flexibility, performance, and interoperability
- reduce the engineering effort involved in developing, transitioning, and operationalizing new analytics
- serve all major constituencies within the network security community, including data scientists, first-tier incident responders, system administrators, and hobbyists
SiLK: The Next Generation?

Mothra **is not** the next version of SiLK

- SiLK’s design philosophy was inspired by UNIX
  - Command-line tools that each focus on doing one thing well
  - Tools are composable into analytics via shell scripting
  - Fixed-length record formats for optimal performance
- With larger, variable-length records, this design can’t scale
  - Solution? Throw more hardware at the problem ("big data")

We view SiLK and Mothra as complementary projects that will be developed in parallel for the foreseeable future

- SiLK still performs well for queries that don’t look beyond layer 4
- Mothra enables more complex analyses at a scale beyond the capability of SiLK’s single-node architecture
Mothra: A Large-Scale Data Processing Platform for Network Security Analysis

Architecture and Design
YAF to SiLK Data Flow

Architecture and Design
YAF to Mothra Data Flow
SiLK vs. Mothra

Mothra departs from SiLK’s UNIX-like design in significant ways:

- **SiLK**
  - Command-line tools, mostly written in C, with some Python
  - Analytics are written as UNIX shell scripts

- **Mothra**
  - Built on Apache Spark, a cluster computing framework
    - Written primary in Scala, which runs on the Java Virtual Machine
    - Language bindings for Java, Python, R, and SQL
    - Runs standalone or on an existing cluster platform (e.g. Hadoop)
    - Mothra’s core libraries are written in Scala and Java
  - Analytics can be written using any language Spark supports
  - A web notebook interface is provided for developing analytics
Platform Languages and Technologies (continued)
Why Spark?

Building Mothra on an established platform like Spark, with its active industry-sponsored open source development community, allows us to focus on components that deliver value to analysts.

The Spark platform:

• enables a degree of scalability not possible with SiLK
• supports higher-level languages for faster development and transition of core functionality and analytics
• provides consistent interfaces to a variety of data sources
• includes libraries for graph analysis and machine learning
• integrates well with other big data platforms and technologies
### Mothra Architecture

**Data Sources**
- YAF
- SiLK

**Analysis Platform**
- Mothra libraries
- Apache Spark Core API
  - Scala
  - Java
  - Python
  - R
  - SQL

**Interfaces**
- Jupyter Notebook
- Exploratory Analysis

**HDFS**
SiLK vs. Mothra Scalability

Query Latency vs. Query Complexity

- Overhead of Spark executor startup, network transfer between cluster nodes, writing temporary files to disk, etc.

- Queries that are faster with SiLK
- Queries that are faster with Mothra
- Queries SiLK can’t handle due to scale / complexity, but are possible with Mothra
User Interfaces

Mothra uses Jupyter Notebook as an exploratory analysis UI:

- Rich web-based interface
  - Input cells for developing and executing code
  - Output cells display analysis results, including visualizations
  - Markdown for annotations with rich text
- Simple sharing and publishing of analytics and results
- Less daunting for novices to learn than the UNIX command-line

For CLI fans, `jupyter-console` and `spark-shell` are available
Architecture and Design

Jupyter Notebook

3.3.4 Filtering

Note that there are a lot of null values in the SSL columns. Let’s filter those out and display only the flows that have SSL fields:

```python
In [11]: https_flows.filter($'sslCertificate'.isNotNull()).show()
```

3.3.5 Aggregation

The syntax for aggregate queries is relatively simple. In this query, we’ll group the `https_flows` DataFrame by destination IP address with `https_flows.groupby($'dip'`). Calculating the average packets and bytes with `.avg("packets", "bytes")`, and sort the resulting rows by the average byte count with `.sort($"avg(bytes)").desc`.

```python
In [12]: https_flows.groupby($'dip').avg("packets", "bytes").sort($"avg(bytes)").desc().show()
```
Mothra: A Large-Scale Data Processing Platform for Network Security Analysis

Demonstration
Field Specifier Syntax

IPFIX fields are specified using strings of the following format:

```
[path][/][operator]name[:format][=alias]
```

where:

- **path** (optional) is a path to the desired information element
  - paths are made up of template names, delimited by / characters
  - if path is empty, the field specifier will look for the given element in all top-level IPFIX records
- **operator** (optional) is a character indicating how the information element should be treated
  - currently, the only operator is @, indicating that the IE is a basicList field
- **name** (required) is the name of the IPFIX information element
- **format** (optional) is a string indicating how the field should be formatted in the data frame
  - current formats are:
    - str – format IE as a string
    - iso8601 – format IE as an ISO-8601 date string
    - base64 – format IE as a base64-encoded string
- **alias** (optional) is a string to be used for the data frame column name instead of the IPFIX IE name
  - if alias is unspecified, the column name will default to the IPFIX IE name
Field Spec Examples

• `flowStartMilliseconds`
  • the flow start time in milliseconds at the top level of any IPFIX record

• `flowStartMilliseconds:iso8601`
  • same, but formatted as an ISO-8601 string

• `flow_total_ip6/flowStartMilliseconds:iso8601`
  • same, but only if the top-level record matches the IPv6 template

• `flow_total_ip6/flowStartMilliseconds:iso8601=stime`
  • same, but rename the field to stime

• `/http/@httpUserAgent`
  • the HTTP user agent list within the http template

• `/ssl_cert_full/@sslCertificate:base64`
  • a list of SSL certificates in the flow, each base64-encoded
Spark DataFrames

From the Spark documentation:

• "A DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood."

To build a DataFrame in Mothra with the Scala language interface:

In [4]: `val input_df = spark_ipfix.load(sqlContext, input_data, all_fields)`

To count the number of records:

In [5]: `input_df.count()`

Out[5]: 30299
Queries

Spark DataFrame API maps reasonably well to \texttt{rw*} commands:

- Filtering (\texttt{rwfilter})

```python
In [6]: var https_flows = input_df.filter($"dport" === 443)
```

- Syntax is similar with the Python API:

```python
In [48]: https_flows = df.filter(df["sport"] == 443)
https_flows.show()
```
Queries

- Column selection & display (\texttt{rwcut})

```python
In [8]: https_flows = https_flows.select(
    "sip", "dip", "sport", "dport",
    "proto", "packets", "bytes", "sslCipher", "sslCertificate")
https_flows.show()
```

<table>
<thead>
<tr>
<th>sip</th>
<th>dip</th>
<th>sport</th>
<th>dport</th>
<th>proto</th>
<th>packets</th>
<th>bytes</th>
<th>sslCipher</th>
<th>sslCertificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.202.80</td>
<td>192.168.28.254</td>
<td>53708</td>
<td>443</td>
<td>6</td>
<td>2</td>
<td>84</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>192.168.202.80</td>
<td>192.168.28.254</td>
<td>53709</td>
<td>443</td>
<td>6</td>
<td>2</td>
<td>84</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>192.168.202.80</td>
<td>192.168.28.254</td>
<td>53710</td>
<td>443</td>
<td>6</td>
<td>2</td>
<td>84</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>192.168.202.80</td>
<td>192.168.28.254</td>
<td>53711</td>
<td>443</td>
<td>6</td>
<td>2</td>
<td>84</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>192.168.202.83</td>
<td>192.168.206.44</td>
<td>58624</td>
<td>443</td>
<td>6</td>
<td>1</td>
<td>60</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>192.168.202.83</td>
<td>192.168.206.44</td>
<td>58628</td>
<td>443</td>
<td>6</td>
<td>1</td>
<td>60</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>192.168.202.80</td>
<td>192.168.28.103</td>
<td>54128</td>
<td>443</td>
<td>6</td>
<td>5</td>
<td>294</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
| 192.168.202.80 | 192.168.28.103  | 54151 | 443   | 6     | 6       | 412   | [22, 19, 10, 102, ...] | [MD8BPzA/AQgCCQA/...]
| 192.168.202.80 | 192.168.28.103  | 54153 | 443   | 6     | 12      | 1036  | [5, 4, 65664, 2, ...] | [MD8BPzA/AQgCCQA/...]
| 192.168.202.76 | 192.168.22.254  | 61187 | 443   | 6     | 11      | 1359  | [4, 5, 47, 51, 50...] | [MD8BPzA/ARI/AwIB...]|
Queries (continued)

• Sorting:

In [10]: https_flows.sort("bytes".desc).show()

• Aggregation:

In [12]: https_flows.groupBy("dip").avg("packets", "bytes")
            .sort("avg(bytes)".desc).show()

<table>
<thead>
<tr>
<th>dip</th>
<th>avg(packets)</th>
<th>avg(bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.28.253</td>
<td>6.0</td>
<td>470.6666666666667</td>
</tr>
<tr>
<td>192.168.28.103</td>
<td>6.0</td>
<td>448.5</td>
</tr>
<tr>
<td>192.168.22.254</td>
<td>4.571428571428571</td>
<td>448.2857142857143</td>
</tr>
<tr>
<td>192.168.4.86</td>
<td>9.0</td>
<td>432.0</td>
</tr>
<tr>
<td>157.55.130.153</td>
<td>3.0</td>
<td>152.0</td>
</tr>
<tr>
<td>65.55.223.14</td>
<td>3.0</td>
<td>152.0</td>
</tr>
</tbody>
</table>
Queries (continued)

- Full SQL syntax

```python
In [17]:
input_df.registerTempTable("df")
sqlContext.sql(""
SELECT dnsQName,
    AVG(packets) AS avg_packets,
    SUM(packets) AS sum_packets,
    AVG(bytes) AS avg_bytes,
    SUM(bytes) AS sum_bytes
FROM df
WHERE dnsQName IS NOT NULL
GROUP BY dnsQName
ORDER BY sum_bytes DESC
""').show()
```

<table>
<thead>
<tr>
<th>dnsQName</th>
<th>avg_packets</th>
<th>sum_packets</th>
<th>avg_bytes</th>
<th>sum_bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>version.bind.</td>
<td>5.372093023255814</td>
<td>231</td>
<td>321.86046511627904</td>
<td>13840</td>
</tr>
<tr>
<td><a href="http://www.apple.com">www.apple.com</a>.</td>
<td>4.0</td>
<td>16</td>
<td>236.0</td>
<td>944</td>
</tr>
<tr>
<td><a href="http://www.squid-cache.org">www.squid-cache.org</a>.</td>
<td>1.0</td>
<td>8</td>
<td>65.0</td>
<td>520</td>
</tr>
<tr>
<td>time.apple.com.</td>
<td>4.0</td>
<td>8</td>
<td>240.0</td>
<td>480</td>
</tr>
<tr>
<td>http.</td>
<td>2.0</td>
<td>6</td>
<td>140.0</td>
<td>420</td>
</tr>
<tr>
<td>44.206.168.192.in...</td>
<td>1.0</td>
<td>3</td>
<td>73.0</td>
<td>219</td>
</tr>
<tr>
<td>fs-1-one.ubuntu.com.</td>
<td>1.0</td>
<td>2</td>
<td>65.0</td>
<td>130</td>
</tr>
</tbody>
</table>
Queries (continued)

- Compound query example
  - Build dataframe of SSL flows with a known bad SSL cert
  - Build dataframe of DNS flows with non-null qname
  - Join two dataframes on the sip field
  - Select the sip, sslCertificate, and dnsQName fields

```python
In [19]:

    (https_flows.filter(array_contains("sslCertificate", bad_ssl_cert))
     .join(dns_flows.filter("dnsQName".isNotNull), "sip")
     .select("sip", "sslCertificate", "dnsQName").show()
```

<table>
<thead>
<tr>
<th>sip</th>
<th>sslCertificate</th>
<th>dnsQName</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.202.80</td>
<td>[MD8BPzA/AQgCCQA/... version.bind.</td>
<td></td>
</tr>
<tr>
<td>192.168.202.80</td>
<td>[MD8BPzA/AQgCCQA/... version.bind.</td>
<td></td>
</tr>
<tr>
<td>192.168.202.80</td>
<td>[MD8BPzA/AQgCCQA/... version.bind.</td>
<td></td>
</tr>
</tbody>
</table>
Mothra: A Large-Scale Data Processing Platform for Network Security Analysis

Future Work
Future Work

On the horizon:

• Streaming ingest from sensors
• Operational analyst console integration
• Simplified deployment and configuration
• Open source release
• Integration of useful components into other FOSS projects
Related Projects

Apache Metron (incubating)
- Sponsored by Hortonworks, Rackspace, Cisco, and others

Apache Spot (incubating):
- Sponsored by Cloudera, Intel, EBay, and others

Similar in scope and scale, different in emphasis and design

As these projects mature and grow in popularity, we may pursue integration opportunities
Questions?

CERT NetSA Tools Home
http://tools.netsa.cert.org

Contact
netsa-help@cert.org
tonyc@cert.org

Mailing list
netsa-tools-discuss@cert.org