Rapid Prototyping of Flow-Based Detection Methods Using Complex Event Processing

Petr Velan, Martin Husák, Daniel Tovarňák
Institute of Computer Science
Masaryk University
Brno, Czech Republic
{velan, husakm, tovarnak}@ics.muni.cz

***Abstract*—Detection of network attacks is the first step to network security. Many different methods for attack detection were proposed in the past. However, descriptions of these methods are often not complete and it is difficult to verify that the actual implementation matches the description. In this demo paper, we propose to use Complex Event Processing (CEP) for developing detection methods based on network flows. By writing the detection methods in an Event Processing Language (EPL), we can address the above-mentioned problems. The SQL-like syntax of most EPLs is easily readable so the detection method is self-documented. Moreover, it is directly executable in the CEP system, which eliminates inconsistencies between documentation and implementation. The demo will show a running example of a multi-stage HTTP brute force attack detection using Esper and its EPL.**

I. Introduction

Most of the current implementations of attack detection methods are written in a programming language. However, it is a difficult task to verify that the implementation was done correctly. It requires a series of tests and even then the tests themselves might be incomplete. Basically, there is no simple way to verify, that the implementation matches the requirements and the description of the given method. Another approach can be used to ensure more "what you see is what you get" result. For example, the widely known NfSen tool has a feature to trigger alerts based on filters. The filter can be a simple query to the underlying flow database, which is easy to read and understand. However, the attack detection methods that can be described in this way is very limited due to the limited power of the filter queries.

The Complex Event Processing (CEP) [1] provide much greater variability than the NfSen filters. The Event Processing Languages (EPL) often build upon SQL standard and enhance it for work with streams of data and time intervals. Moreover, it is possible to easily chain multiple EPL queries to build more complex detection methods if necessary.

The goal of this demo is to show that using EPL for detection methods provides two major benefits: Firstly, it facilitates rapid development of detection methods. Secondly, it allows us easily read and verify what the method does. In this demo, we are going to present a detection method for HTTP brute force attacks, as described in [2].

978-1-5386-3416-5/18/$31.00

II. Demo Components

The source of data for our detection method is network flow monitoring [3]. The flow records are enhanced by information from the application layer and are exported in the IPFIX [4] format to a collector. The collector passes the data to the CEP engine, which processes it based on the EPL description of the detection method. The overview of the architecture is shown in Figure 1.

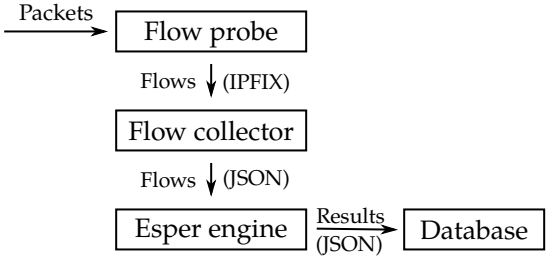

Fig. 1. Demo Architecture Schema

The only requirement on the flow source in our demo is that it must be able to provide host and URL information from HTTP requests. We use Flowmon probe[1] to generate flows one and replay the data in a loop during the demo.

The purpose of the flow collector in this scenario is to receive flow data, convert them and send them to the CEP engine. We use JSON as a universal data exchange format between the flow collector and the CEP engine. The performance of this solution is good enough to handle several thousand flow records per second, which is sufficient to handle the flows from entire campus network of Masaryk University. Should the need arise, a more compact data format could be used to increase the performance. Moreover, with some additional work, it is possible to design the CEP engine to process flows directly. We use IPFIXcol [5] flow collector in our demo.

From a historical perspective, traditional Database Management Systems (DBMSs) oriented on *data sets* were not designed for rapid and continuous updates of individual data items arriving at high velocities required by data-intensive real-time applications in the areas of financial transactions, network monitoring, security monitoring, telecommunications,

[1] https://www.flowmon.com/en/products/flowmon/probe

manufacturing, or sensor networks, and performed very poorly in such scenarios.

As pointed out by Babcock et al. [6], to overcome these limitations, a new class of data management applications emerged. *Data Stream Management Systems (DSMSs)* orient on evaluating *continuous queries*, i.e. queries issued once and then producing results until removed, over *data streams*, i.e. possibly infinite sequences of data elements.

Complex Event Processing (CEP) [1], [7] in general follows the same goals and principles as DSMSs, yet as it is apparent from the term, it is focused on the processing of a very specific type of data elements – *events*, i.e. meaningful occurrences in a particular domain. CEP allows its users to detect complex (composite) events based on expressive queries, e.g. using sequence patterns, temporal constraints, windows, filters, and aggregations [8]. Typically, the complex events can be re-introduced into the processing system as new data streams available for further processing, i.e. it is possible to create new complex events in a hierarchical manner, which results in a powerful data processing paradigm.

Esper[2] is one of the most mature open-source CEP engines. Esper is centered around an SQL-like declarative *Event Processing Language*, which is used to describe continuous queries with the ability to detect quite complex situations.

III. Demo Description

This demo will show a running example of the setup described in the previous section. We will show how the flow data are processed by the flow collector, converted, and passed to Esper. The capabilities of Esper and the EPL will be demonstrated on multi-stage HTTP brute-force attack detection, which is a composition of different EPL queries and is described further in this section.

In our previous work, we studied attacks on HTTP via network traffic analysis [2]. We found two types of attacks on HTTP that could be detected: scanning over HTTP and HTTP brute-forcing. The first attack is similar to well-known TCP SYN scans, but happens on the application layer. The attacker sends the same request to many hosts in the network, e.g., to enumerate vulnerable content management systems. When such a resource is found, the attack may proceed to the second type of attack, brute-forcing of a login page, which can be observed as a series of the same HTTP requests on the same URL. Not only a correlation between the two types of attacks was found, but the HTTP scanning was often preceded by TCP SYN scans. However, we had to implement detection methods and detect each attack phase individually, before proceeding to correlate their results, which was time-consuming. With the CEP-based approach and querying language, we can very easily set up three queries to detect the individual attack stages, and a fourth query, that correlates the outputs of the previous three queries.

We include a sample of a query that can be used to detect HTTP brute-forcing. The example goes as follows:

[2]http://espertech.com/esper/

```
@Name('BruteForce')
SELECT
  ipfix.sourceIPv4Address as Attacker,
  ipfix.destinationIPv4Address as Destination,
  ipfix.HTTPRequestHost as Host,
  ipfix.HTTPRequestURL as URL,
  count(ipfix.sourceIPv4Address) as AtkCount
FROM IPFIX.win:time(1 hour)
WHERE
  ipfix.HTTPRequestURL LIKE '%login%'
  or
  ipfix.HTTPRequestURL LIKE '%admin%'
GROUP BY
  ipfix.sourceIPv4Address,
  ipfix.destinationIPv4Address,
  ipfix.HTTPRequestURL
HAVING count(ipfix.sourceIPv4Address) > 50;
```

As we can see, the query matches the description of the attack we want to detect. Only the flow records containing HTTP request with a specific substring are selected. Subsequently, if there are more such requests from a single source to a single destination in the time window of 1 hour, the source IP address is returned as output.

The next query example illustrates a correlation of outputs of individual detection methods:

```
@Name('Output')
SELECT
  TCPSYNscan.attacker as attacker,
  TCPSYNscan.atkCount as TCPSYNscanCount,
  HTTPscan.atkCount as HTTPscanCount,
  BruteForce.atkCount as BruteForceCount
FROM
  TCPSYNscan.win:time(5 hours),
  HTTPscan.win:time(5 hours),
  BruteForce.win:time(5 hours)
WHERE
  TCPSYNscan.attacker = HTTPscan.attacker
  AND
  TCPSYNscan.attacker = BruteForce.attacker;
```

The query takes outputs of the detection methods as input and searches for IP addresses that caused all of the three attack steps during a time window (5 hours in this example).

IV. Conclusion

We have presented a working concept of network attack detection using complex event processing (CEP) approach. The concept has been demonstrated on a multi-stage HTTP brute-force attack detection, which is a combination of several partial detections. Instructions to build the demo, used software, and all necessary configurations have been made available as open-source at https://github.com/CSIRT-MU/FlowCEP.

Acknowledgment

This research was supported by the Security Research Programme of the Czech Republic 2015 - 2020 (BV III / 1 VS) granted by the Ministry of the Interior of the Czech Republic under No. VI20162019029 The Sharing and analysis of security events in the Czech Republic.

References

[1] O. Etzion and P. Niblett, *Event Processing in Action*, 1st ed. Manning Publications Co., 2010.

[2] M. Husák, P. Velan, and J. Vykopal, "Security monitoring of http traffic using extended flows," in *2015 10th International Conference on Availability, Reliability and Security*, Aug 2015, pp. 258–265.

[3] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2037–2064, Fourthquarter 2014.

[4] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011 (Internet Standard), RFC Editor, Fremont, CA, USA, pp. 1–76, Sep. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7011.txt

[5] P. Velan and R. Krejčí, "Flow Information Storage Assessment Using IPFIXcol," in *Dependable Networks and Services*, ser. Lecture Notes in Computer Science, vol. 7279. Heidelberg: Springer Berlin Heidelberg, 2012, pp. 155–158.

[6] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '02. ACM, 2002, pp. 1–16.

[7] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[8] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," *ACM Comput. Surv.*, vol. 44, no. 3, pp. 15:1–15:62, 2012.