MalFIX: Using IPFIX for Scaling Threat Detection to High Data Rates

Gabriel Paradzik[‡§], Benjamin Steinert[‡§], Janik Steegmüller, Michael Menth[‡]
[‡] University of Tübingen, Chair of Communication Networks, Tübingen, Germany
[§] University of Tübingen, Zentrum für Datenverarbeitung, Tübingen, Germany

***Abstract*—Threat intelligence feeds provide up-to-date information about threat indicators, i.e., IP addresses, hostnames, etc. This information can be used to identify potentially malicious actors by scanning network traffic. In this paper, we present a high-performance architecture for threat detection that leverages openly available threat intelligence feeds. For that purpose, the open-source tool Maltrail has been modified to make it horizontally scalable and to handle IPFIX flow data. Maltrail was adapted to process IPFIX as input and generate IPFIX-compatible output that includes information about detected threats. These threats are then ingested into Apache Kafka, enabling further analysis and integration with other tools. Benchmark results highlight the scalability of this approach, with a peak processing speed of 300,000 flows per second on 32 CPU cores.**

I. Introduction

The use of openly available threat intelligence feeds allows network operators to identify bad actors before any potential damage has been done. Recognizing these threats is crucial for taking preventive action, i.e., blocking malicious traffic, or notifying users who were exposed to malicious actors. There are many software tools that address this problem, but they have various shortcomings.

One of the big issues is scalability. Many tools work on raw packet captures which is not feasible for high traffic rates. The problem becomes more serious the more locations on the network are monitored. Monitoring raw traffic also makes it difficult to distribute the threat analysis load onto multiple machines.

Another issue is lack of modularity and extensibility. A modular threat detection architecture ensures that any part can be upgraded if it becomes a bottleneck. Extensibility is essential for further processing of detected threat events or integrating the architecture into a larger network monitoring system.

This paper presents a modular threat detection pipeline using the open-source tool *Maltrail* [4]. We modified Maltrail to process IPFIX flows and achieve a scan rate of up to 300k flows per second. Detected threats are published using the event streaming principle and thus can be easily integrated with other analysis tools adhering to event streaming.

The usefulness of IPFIX for traffic monitoring in general is discussed in various papers. Hofstede et al. [11] provide an overview of all stages of a network monitoring setup using IPFIX. They mention a reduction in traffic to analyze compared to packet-based monitoring of $1/2000$ and the potential in various use cases including intrusion detection, due to the extensibility of the IPFIX protocol.

The rest of the paper is structured as follows. Section II discusses the relevant technical background. In section III the MalFIX architecture is proposed. Afterwards a performance evaluation is conducted in section IV.

This work was supported by the bwNET2.0 project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

II. Technical Background

This section introduces relevant concepts of threat intelligence feeds, event streaming, and network flow generation and collection.

A. Threat Intelligence Feeds

Threat intelligence feeds can provide insights into the reputation of IP addresses and domains to help identify known malicious actors on the Internet. These feeds are available in various formats, ranging from unstructured text-based indicators of compromise (IoCs) and IP blocklists to highly structured, machine-readable threat sharing standards such as STIX and TAXII [5]. The publication of such feeds is done by security vendors, private companies, or open source communities. In some cases, the processes used to generate these feeds are transparent and well-documented (e.g., CESNET NERD [10]), while in others the methodologies remain unclear. Additionally, some providers aggregate data from multiple feeds to offer a more comprehensive threat picture, e.g., IPsum [3]. Although public threat data is abundant, its reliability varies, and matching every network packet against all available feeds is computationally impractical.

B. IPFIX

IPFIX [9] is a protocol for aggregating packets into *flows*. A *flow* represents a completed communication between two participants on the network. It is usually characterized by the 5-tuple (src IP, dest IP, src port, dest port, L4 protocol) and contains metadata about the flow. This includes: Start and end timestamps, octet count, packet count, etc.

Each flow record consists of multiple Information Elements (IEs). The recorded flow metrics are saved into IEs which describe the type of data they hold. The actual payload information of the communication is usually discarded.

The IPFIX standard provides a way to include arbitrary data points via custom IEs. Flow generating devices can use these fields to embed data points which are not provided by the IPFIX standard. Examples of this are the set of observed TCP flags, OS/application fingerprinting, largest recorded packet size, etc.

The structure of flow records is defined by IPFIX templates. Each flow may contain a different set of IEs depending on the protocols recorded. Templates are not predefined by the IPFIX standard, but are dynamically created by the sender. The set of all used templates is periodically sent to the recipient.

Devices generating IPFIX data are called *flow meters*. A network can contain multiple flow meters installed at strategic points, e.g., core routers, edge routers. All IPFIX records generated by flow meters are sent to *flow collectors* which store or analyze flow information.

C. Event Streaming

Event Streaming is a method for conveying a continuous flow of data. Each data point in this concept is called an *event*. Events are generated by one or many producers and sent to the *event broker*. Each event is published in a *topic*. The purpose of topics is to group events of the same type. This allows an event broker to handle multiple event types. Besides producers, there are also *consumers* which subscribe to topics. After receiving an event for a topic, the broker forwards it to one or more consumers subscribed to that topic. Prominent event brokers are Apache Kafka [1] and RabbitMQ [7] which are widely used.

III. MalFIX Architecture

Maltrail is an open-source threat detection system written in Python. It listens to network traffic on an interface and searches for signature-based and heuristics-based threat indicators. To identify malicious traffic, Maltrail keeps an updated list of threat indicators. They include IP addresses, hostnames, and payload signatures from openly available threat intelligence feeds. Upon detecting a potential threat, a log message is written to a file recording the event. A filtering action is not taken as Maltrail only monitors network traffic passively. Threat events are displayed to the user via a web interface which lists the recorded threats in a table. Fields include the recorded time, the offending IP address/port, trail information, i.e., the malicious signature, etc.

The primary use case for Maltrail is for monitoring end systems or traffic on switches with relatively low traffic rates. Because the capture library *libpcap* is used, monitoring high traffic rates is not feasible without packet drops. Maltrail records detected threats as log messages in an append-only text file. As text files are not indexed, this makes search operations expensive. Along with a high detection rate, this slows down the subsequent analysis step.

We propose the *MalFIX* architecture shown in Figure 1. It employs several open-source tools as well as Maltrail. Changes to the Maltrail code base have been made minimally invasive, so that future upstream code changes can be easily merged. We call our modified version of Maltrail *MalFIX*[1]. The changes to the input and output of Maltail are discussed in the following.

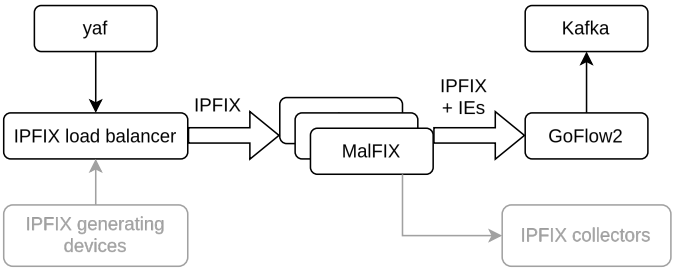

Fig. 1. Architecture of the MalFIX threat detection pipeline.

A. Input Adaptations

Instead of raw traffic captures, MalFIX receives IPFIX as input. IPFIX flow records are then rewritten into IP packets and fed into to Maltrail's threat detection engine. This allows the Maltrail code base to remain unchanged and upstream code changes to be easily merged.

Higher traffic rates can be monitored with the same load by using IPFIX instead of raw traffic captures. To generate IPFIX, we employ *yaf* [8] which is an open-source flow meter developed by the Network Situational Awareness (NetSA) group at the CERT coordination center. It can be configured to use the PF_RING™ [6] library for high-speed packet processing to handle incoming network traffic. By using an alternative packet path through the kernel, it creates fewer internal copies while forwarding the packet to the application, thus increasing performance.

Alternatively, many network devices support exporting IPFIX data which can also be used as input for MalFIX. To make MalFIX more scalable, we employ an IPFIX load balancer. This allows multiple instances of MalFIX to run. IPFIX data is then be passed around in a round-robin fashion.

B. Output Adaptations

To solve the shortcoming of the file output format, we use the custom IEs feature of IPFIX. When a threat is detected, relevant information is attached via a custom IE to the original IPFIX record. This conforms to the IPFIX standard and thus creates an output which is readable by any IPFIX-compatible flow collector.

The goal is to forward detected threats via event streaming. Apache Kafka is used for the event streaming component, but it cannot act directly as a flow collector. This is because Kafka does not support the IPFIX protocol. To overcome this issue, we first convert it to Protobuf by using *GoFlow2* [2]. Protobuf is a binary serialization format developed by Google. GoFlow2 uses Protobuf with a statically defined IPFIX template to encode incoming flows. As a result, Kafka receives all IPFIX flows in the same format.

MalFIX supports two modes of operation. In *Pipeline mode* all incoming flows are forwarded to the receiving application.

[1]https://github.com/uni-tue-kn/MalFIX

Malicious IPFIX flows are augmented by adding threat information. In *Alert-Only mode* MalFIX only malicious flows are forwarded while non-malicious flows are discarded.

C. Deep Packet Inspection

By switching from packet-based to flow-based threat detection, we lose payload information. Threat detection using IP/port information is still possible, but searching for malicious signatures in the packet payload is no longer feasible. To retain some of this original Maltrail functionality, the deep packet inspection (DPI) functionality of yaf is used. Yaf can scan packet payloads for defined patterns and store matches in custom IEs. Examples of detected fields are DNS query name/response, HTTP header fields, SSL certificate information, etc.

MalFIX checks the custom IEs of DNS queries and searches its internal threat database for the requested DNS name. Upon detection of a malicious DNS request, MalFIX records the threat event in the outgoing IPFIX flow record via a custom IE, analogous to detecting a malicious IP address.

IV. Performance Evaluation

This section investigates the performance evaluation of MalFIX. First, the methodology is introduced, followed by the presentation of the results.

A. Testbed and Methodology

Performance tests were conducted on virtual machines (VMs) running on a host with an AMD EPYC 7543 32-core processor. The setup consists of three main components running on separate VMs. First, there is a generation process for IPFIX records, which are possibly annotated with DNS information. Second, a VM which runs all MalFIX instances. Finally, a flow collection process is also part of the system. The virtual machine running MalFIX has Ubuntu 22.04 with Linux 5.15 installed and was assigned 32 CPU threads and 16 GB memory.

For the performance evaluation, MalFIX is fed with artificially generated IPFIX records. These are generated by a load generator and sent to MalFIX via TCP. The processing speed of MalFIX limits the transmission rate at which flow records are sent over TCP, and the load generator only generates new IPFIX records when new ones are transmitted via TCP. This means that the rate at which IPFIX records are generated and processed is the same. If multiple MalFIX instances are used each is fed with IPFIX records by the load generator via a TCP connection. In a practical deployment, this can be realised using a load balancer for IPFIX records.

B. Results

The load generator generates IPFIX records for malicious and non-malicious flows, some of which are annotated with DNS information. To assess how the share of malicious flows affects MalFIX's load, we varied their fraction in the performance study.

- *100% Malicious* consists of purely malicious flow records with a malicious IP address.

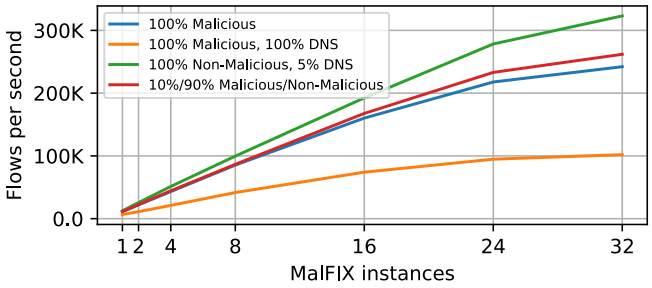

Fig. 2. MalFIX performance measurements with different traffic patterns.

- *100% Malicious, 100% DNS* consists of purely malicious flow records with a malicious DNS query inside a custom IE.
- *100% Non-Malicious, 5% DNS* consists of purely non-malicious flow records while 5% of overall flows have some DNS name attached.
- *10%/90% Malicious/Non-Malicious* consists of 10% malicious flows and 90% non-malicious flows. The 10% of malicious traffic is made up of 50% malicious IP addresses and 50% malicious DNS queries.

MalFIX was configured to be in *Alert-Only mode*, forwarding only detected threats.

The results are shown in Figure 2. The highest flow processing rate was measured for purely non-malicious traffic. In this case, no outbound IPFIX flows need to be created, which is a costly operation. The traffic with 10% malicious traffic had a slightly lower processing rate as some IPFIX flows were sent by MalFIX. The worst result was measured for both traffic patterns with 100% malicious flows. Between the two, the processing rate dropped by more than 50% when DNS names had to be checked. The processing speed grows linearly with an increasing number of MalFIX instances, which demonstrates the horizontal scalability of MalFIX. For 32 instances, we measured a slightly lower processing rate than expected from a linear increase. This is due to the virtual machine reaching the limit of 32 available CPU cores.

References

[1] "Apache Kafka." [Online]. Available: https://kafka.apache.org/
[2] "GoFlow2." [Online]. Available: https://github.com/netsampler/goflow2
[3] "IPsum – Daily Threat Intelligence Feed of Bad IPs." [Online]. Available: https://github.com/stamparm/ipsum
[4] "Maltrail." [Online]. Available: https://github.com/stamparm/maltrail
[5] "OASIS Cyber Threat Intelligence Technical Committee - STIX and TAXII." [Online]. Available: https://oasis-open.github.io/cti-documentation/
[6] "PF_RING." [Online]. Available: https://www.ntop.org/products/packet-capture/pf_ring/
[7] "RabbitMQ." [Online]. Available: https://www.rabbitmq.com/
[8] "yaf." [Online]. Available: https://tools.netsa.cert.org/yaf2/index.html
[9] P. Aitken, B. Claise, and B. Trammell, "RFC7011: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," Sep. 2013.
[10] V. Bartoš, "NERD: Network Entity Reputation Database," in *International Conference on Availability, Reliability and Security*, 2019.
[11] 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, 2014.