Big Data Security Analysis with TARZAN Platform

Marek Rychlý* and Ondřej Ryšavý

Brno University of Technology, Faculty of Information Technology, Department of Information Systems, IT4Innovations Centre of Excellence, Brno, Czech Republic
E-mail: rychly@fit.vutbr.cz; rysavy@fit.vutbr.cz
**Corresponding Author*

Received 20 November 2017; Accepted 26 September 2018;
Publication 06 November 2018

Abstract

The TARZAN platform is an integrated platform for analysis of digital data from security incidents. The platform serves primarily as a middleware between data sources and data processing applications, however, it also provides several supporting services and a runtime environment for the applications. The supporting services, such as a data storage, a resource and application registry, a synchronization service, and a distributed computing platform, are utilized by the TARZAN applications for various security-oriented analyses on the integrated data ranging from an IT security incident detection to inference analyses of data from social networks or crypto-currency transactions. To cope with a large amount of distributed data, both streamed in real-time and stored, and for the need of a large scale distributed computing, the platform has been designed as a big data processing system ensuring reliable, scalable, and cost-effective solution. The platform is demonstrated on the case of a security analysis of network traffic.

Keywords: Security, Big data, Framework.

1 Introduction

The abundance of data sources and the exponential growth in the volume they produce represents new challenges for many traditional ICT fields.

Journal of Cyber Security and Mobility, Vol. 8_2, 165–188. *River Publishers*
doi: 10.13052/jcsm2245-1439.822
This is an Open Access publication. © 2018 *the Author(s). All rights reserved.*

Digital forensics and security incident analysis is not an exception. Every day, security analysts and investigators face the problem of insufficient tool support. The roots of this problem lie in the fact that this dramatic change in the heterogeneity and volume of data makes the existing methods obsolete.

Traditional workflow of digital forensic consists of the well-defined procedure of data identification, acquisition, preservation, analysis, and reporting. This workflow was devised and refined in the 1990s when the environment regarding computing technology and software was rather uniform. Also, the amount of data that needs to be processed was from our perspective rather small. For most cases, it was possible to perform all above-mentioned steps using a single forensic workstation. Because of the rapid technology advances in the ICT, this is no longer true. Not only the increasing amount of data caused by the drop of storage cost and dissemination of broadband connectivity represents the challenge for digital forensics. Often even the data acquisition phase is difficult to achieve with the existing tools and considering the usual methods of creating the forensic image of the disk drive. Completing this operation for nowadays common terabyte hard drive lasts several hours.

Investigators also need to face the problem of high diversity of computing devices. Smartphones, tablets and other connected smart devices massively penetrate the market. Cloud services are another emerging technology that changes the requirements on the digital forensics methods. All of this means that classical approach represented by well-defined workflow and considering the use of a single forensics workstation cannot meet the current demands. In many cases, the amount of data that to be processed exceeds several terabytes. Also, some forms of cyber crime comprise of the combination of several sophisticated techniques, and for their investigation, it is necessary to process and correlate information from several big datasets. To cope with this problem, several researchers suggested to apply big data approach, e.g., [13], and this field has become an active research area.

In this paper, an integrated platform for analysis of digital data from security incidents (a TARZAN platform) is proposed to address the issues mentioned above. The platform allows to gather, store, and process digital forensic data as big data to perform various security-oriented analyses that range from an IT security incident detection to inference analyses of data from social networks or crypto-currency transactions.

The paper is organized as follows. Section 2 discusses related work on data security analysis and processing platforms. In Section 3, we provide a case study of a PCAP analysis tool utilizing the proposed platform for real-time security analysis of network traffic. Section 4 introduces the TARZAN

platform and describes its architecture and core services. The results of the case study implementation on the platform are discussed in Section 5. Finally, we draw conclusions in Section 7.

2 Related Work

Several approaches were already proposed to perform security, forensic, and inference analyses. Because conventional technologies are not always adequate to support long-term, large-scale analytics [9], big data approaches to the digital forensics started to emerge addressing their own challenges (see [13, 16]). However, the most of the existing approaches focused on particular selected topics of IT security, related often only to networking security, rather than providing a general framework to support and integrate various forensic data to analyse them and their inferences in a broader context, such as in the cases of [21] and [22].

In [21], a digital forensic data reduction process were proposed based on a selective imaging, to speed up the forensic process by locating evidences, or by providing examiners with a quick understanding of the data to enable a better focus for full analysis (e.g., for a cross-device or cross-case analysis). Although the proposed process is general enough to support the examination of various types of the stored big data, it is not designed for custom autonomous big data analyses.

Feature Collection and Correlation Engine (FCCE, [22]) was introduced to find correlations across a diverse set of data types spanning over large time windows with very small latency and with minimal access to raw data. The engine entailed a complete framework for ingesting, aggregating, storing, as well as retrieving big data, by implementing feature extraction, aggregation, storage, and retrieval APIs, respectively. It was applied in IT security to detect fluxing domain names and identify persistent threat infections. However, the engine did not provide an implementation platform to build system utilizing the implemented APIs.

Network forensic analysis, which is the subject of the case study presented in this paper to demonstrate the TARZAN platform (see Section 3), comprises of methods for capturing, collecting and analysing network data for information gathering, evidence identification, or security incident investigation. A new generation of Internet services opens space for new cybercrime activities. Security analyst and Law Enforcement Agency officers have to act accordingly to detect unlawful or unauthorized activities efficiently. The investigation is not possible without the tool support. While technology

advances provide hardware technology able to capture communication at speeds that match current wire speed the software equipment for analysis of captured traffic has difficulties with packet traces of several gigabytes.

Network forensic analysis methods were implemented in various tools. General purpose tools include network analysers (Wireshark, TCP dump), IDS systems (Snort, Bro), fingerprinting tools (Nmap, p0f), and tools to identify and analyse security threats.

Tools dedicated to network forensic analysis implement specific features to aid investigation process. They can capture an entire network traffic and allow an investigator to analyse it and reconstruct the communication. Several widely used open source tools exist. In the following, we briefly overview three freely available tools that employ the typical features. Survey of network forensic tools can be found in [19].

PyFlag is a general purpose forensic package which can be used as disk forensics, memory forensics, and network forensics tool. This tool was developed by M. Cohen of Australian Federal Police in 2005 [10]. PyFlag is designed around the Virtual File System concept. For each supported data source a specific loader is implemented. To deal with PCAP files, the PCAP filesystem loader opens PCAP file, parses and dissects individual packets up to lower layer protocols, collects related TCP packets into streams and finally applies higher level protocol dissectors. A forensic investigator is usually interested in high-level information that can be extracted from the communication. PyFlag enables to reassemble the content of communication, e.g., web pages, email conversation, etc.

Network Miner[1] is an open source tool that integrates packet sniffer and higher-layer protocol analysers into a tool for passive network traffic monitoring and analysis. Because captured traffic can be processed in the same way, Network Miner is also a valuable tool for network forensics analysis. Network Miner offers several useful features, such as the possibility of operating system identification, traffic classification, and reassembling the transferred files for HTTP, FTP, TFTP and SMB protocols.

Xplico[2] is a modular tool aimed at the reconstruction of the data content carried in the network traffic. The software consists of the input module handling the loading source data, decoding module equipped with protocol dissectors for decoding the traffic and exporting the content, and the output module organizing decoded data and presenting them to the user. Contrary to

[1] http://www.netresec.com/?page=NetworkMiner
[2] https://github.com/xplico/xplico

PyFlag and NetworkMiner, Xplico is not a typical desktop application but it is deployed as a server service with the web-based interface. The authors claim the possibility to analyze large PCAP files of many gigabytes. Because the Xplico design is a classical client-server architecture, the performance of the tool is limited by the hardware configuration of the server running the Xplico backend.

To analyse the network traffic as big data, a scalable internet traffic analysis system was presented in [17]. The system, which was able to process multi-terabytes libpcap dump files, utilized Apache Spark for data processing to analyse captured transmitted data and protocol fields. Unfortunately, the system did not allow to integrate non-network data and perform the analyses of the network data in broader contexts.

Another approaches to the network traffic security big data analysis were presented in [8, 14, 20, 24]. Apache Metron [5] and Apache Spot [7] projects are more interesting. They try to form general frameworks for security analyses of IT threats, secondary processing also firewall and application logs, emails, intrusion-detection reports, etc. However, analogously to the first case, also the all of these approaches were focused primarily and narrowly on the network data and unable to find correlations with other forensic data or to provide a comprehensive platform for big data forensics.

3 Case Study: PCAP Analysis

Digital investigators process network traffic as a source of evidence in many types of computer crimes. Captured traffic can be analyzed to obtain the content and also to show the actions taken by the offender. Network traffic can also be important for corroborating evidence. Obtaining network traffic as a source of evidence is usually more complicated than other digital evidence. Transmitted data are only available on the network device for a limited amount of time. Inappropriate collection method can lead to data corruption or incomplete capture. As messages exchanged between applications are segmented into many pieces, it is important to gather all relevant packets and be able to combine them again into data streams. When collecting data on shared links, there may be a huge amount of traffic from which only a fraction is relevant to the investigation. Moreover, many different protocols are in use which requires applying corresponding decoding algorithms. Although existing tools for information security can be adapted for a forensics investigation, they usually lack features for evidence collecting and preservation. For the forensic investigation, there are two important activities, namely examination

and analysis [11]. The examination is characterized by the mostly automatic data processing that ends with a collection of relevant data extracted from the data source. The analysis follows examination, and it is often a manual or more interactive activity that interprets the significance and meaning of the extracted data. Also, data correlation, finding links and patterns in the extracted data is the desired result of the analysis.

From the examination viewpoint, the important features of network forensic tools are as follows:

- *Flow reconstruction.* Because network conversation is split into many packets exchanged by communicating applications, the first step of data examination is to combine these fragments to form flows again.
- *Protocol identification.* Network communication is governed by protocols. There are many protocols in the Internet communication. The ability to identify which protocol was used to data exchange is crucial for further processing. Protocol identification is difficult for encrypted traffic where traditional pattern based methods may be less accurate results.
- *Protocol decoding.* To understand the communication we often need to extract data from protocol header fields and data payload. Network forensic tools support a wide variety of protocols. Sometimes these decoders can identify anomalous packets that do not conform to the protocol specification.
- *Data reduction.* Not all data needs to be analysed. Data reduction can filter out uninterested data. The filter applied depends on the information obtained from the protocol decoding step. We can be, for instance, interested only in Web traffic.
- *Data recovery.* If communication is not encrypted, the communication payload is visible. This gives us the possibility to recover digital objects from the communication such as web pages, images, e-mail messages.
- *Pattern search.* The common investigative approach is to search for occurrences of known patterns, e.g., email addresses, keywords, etc. Pattern search in network traffic needs to consider specifics of various protocols, such as encoding, compressing, etc.

Forensic data analysis can involve different methods and procedures. The following techniques are commonly applied:

- Developing the *timeline* from significant events offers investigators a high-level view on the extracted data. Different kinds of communication can contribute to timeline by various events, such as e-mail delivery, web search, file download, etc.

- The *temporal analysis* aims to identify patterns or anomalies that are often processed by the further and deeper analysis. For instance, we are seeking for the periods of peak data transfer or occurrences of an unusual protocol.
- The *relation analysis* provides links among different entities. Relations can be analysed on different layers, linking devices, services, or end-users.
- *Classification* methods assign extracted data to different classes according to the predefined criteria, such as system traffic, web traffic, suspicious traffic, malware related traffic, etc.
- *Clustering* techniques can deal with a lot of entities by grouping them according to some essential characteristics. Often these methods do not require learning and thus are easily applicable.
- *Correlation* is a statistical technique that can identify the relation between different entities. It is, for instance, possible to identify the same activity recorded in various data sources.

Digital investigation is a time-consuming and labour-intensive process. Thus, there is a strong emphasis on using tools that can reduce the examination time and improve the automation of analysis activities. In the next section, we will show, how the proposed platform can achieve both requirements. First, examination time can be reduced by allocating more computation nodes. Second, some analysis can be automated by applying machine learning algorithms.

The complex PCAP analysis requires processing of a huge amount of data. The processing must be done both in real-time to detect security incidents or to perform security audits, and later on large stored datasets to answer queries of an analyst. As the such processing is difficult to do by conventional centralized approaches in highly scalable, high-throughput, and fault-tolerant way [9], the PCAP analysis tool will be implemented on the TARZAN platform.

4 The TARZAN Platform

To ensure communication of TARZAN applications and provide them with required services and a runtime environment, the TARZAN platform consists of three core components, namely, *Platform Bus* which implements a distributed communication bus for the applications and the components, *Platform Storage* which provides a distributed storage service (NoSQL databases, distributed filesystem, resource registries, etc.), and *Platform Computation* component to provide the runtime environment for distributed computation tasks of TARZAN applications.

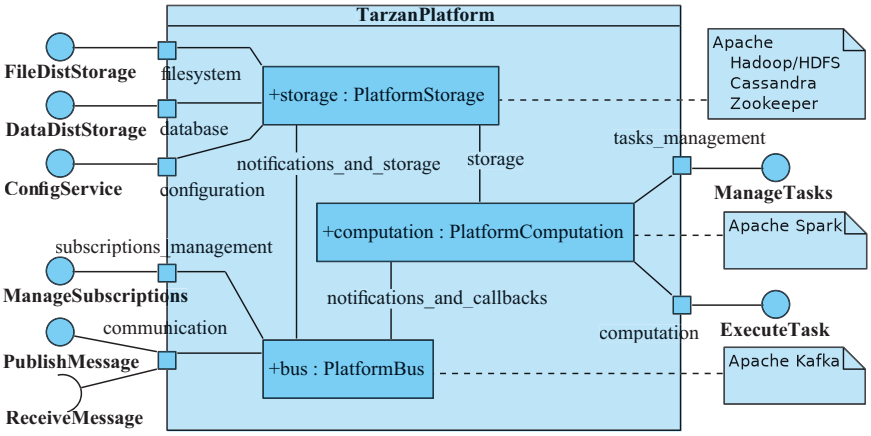


Figure 1 The TARZAN platform architecture.

In Figure 1, architecture of the TARZAN platform is modelled in an UML composite structure diagram. Each of the three core components provides its services to TARZAN applications by the platform's external interfaces. Moreover, the components communicate and cooperate inside the platform. The individual components are described in the following sections.

4.1 Platform Bus

The main goal of the platform bus core component is to enable asynchronous communication of other TARZAN components. More specifically, the platform bus implements a *publisher-subscriber* communication model based on *message queues*. A client is able to publish messages to particular topics acting as a producer, or to subscribe to receive messages of particular topics as a consumer (see the corresponding interfaces in Figure 1). The platform bus guarantees delivery of the published messages to their appropriate consumers.

The communication via the bus is utilized by both external TARZAN applications and the core TARZAN components. In the first case, the applications can connect themselves to various data sources to ingest sensor data, events, logs, etc.; interconnect their components into data processing topologies to perform data parsing, normalizing, validating, marking, enrichment, etc.; and consume or feed data from/into the platform storage components. In the second case, the platform bus helps the other core components to send/receive their data, for example, to store the transmitted data into the platform storage and deliver the storage update notifications, or

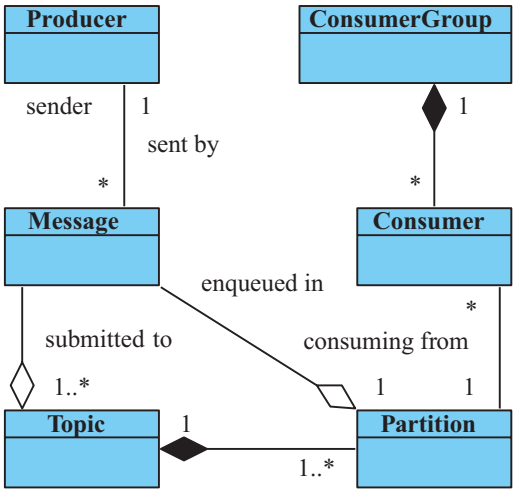


Figure 2 A conceptual model of basic entities in Apache Kafka.

to deliver input data and pass output data of tasks of the platform computation including callbacks.

To achieve high-throughput message passing in highly scalable distributed environments, the platform bus is based on *Apache Kafka* [4]. In Kafka, messages are communicated in topics. Each topic, as a general category of particular messages, consists of multiple partitions (queues). A producer submits a message to a particular topic (or topics) where in each topic, the message is assigned to a particular single partition (automatically for load-balancing or as required by the producer). A consumer can belong to a particular consumer group and subscribes to one or more topics. In each of the subscribed topics, the consumer has assigned particular partition for exclusive reception. For relationships of these concepts, see Figure 2.

In TARZAN, Kafka's concepts of a message, topic, partition, producer, consumer, and consumer group are utilized for consuming data sources and communication with computation tasks as follows.

4.1.1 Broadcasting from data sources

A data source (producer) submits data (a message) of a particular type (topic) under the data source's identification (partition). A subscriber (consumer) listens to a particular topic and a particular partition, that is for messages of the particular type from the particular data source. A message will be received by (broadcasted to) all subscribed consumers in different consumer groups.

- Messages = data produced by the sources.
- Topics = individual data source types (e.g., PCAP).
- Partitions = particular data sources (e.g., a sensor monitoring a network traffic on a particular network interface).
- Consumer groups = subscribers for data produced by a particular data source (e.g., a component for processing/analysing/storing PCAPs).

4.1.2 Load-balancing of data processing tasks

A client (producer) submits a task invocation (message) to a particular service (topic) without any partition (it will be assigned automatically by Kafka for load-balancing). In the case of a request-reply task invocation, the message should contain also the client's identifier which will be utilized for the callback (a particular partition name in "callback" topic).

- Messages = task invocations including data payloads and callback addresses if needed.
- Topics = names of individual services (e.g., PCAP Analyzer).
- Partitions = individual instances of a particular service (e.g., a particular process of the PCAP Analyzer).
- Consumer groups = single-member groups representing the instances as above.

4.1.3 Delivery of the Tasks' Replies

A particular service task instance (producer) submits a reply/result (message) to the previously received task invocation as a callback. The reply (message) will be delivered to a particular client who sent the task invocation (to his partition in "callback" topic).

- Messages = replies/results to the previously submitted task invocations.
- Topics = a single topic with name "callback" only.
- Partitions = one partition for each individual client.
- Consumer groups = single-member groups representing the clients as above.

4.2 Platform Storage

While the platform bus described in the previous section is necessary for data processing, the platform storage implements the data persistence in distributed environments. The distributed data storage is the necessary requirement of distributed data processing to be able to scatter and deliver data across

processing nodes. Three types of data storage services are supported: a distributed filesystem, a distributed database, and a distributed and synchronized configuration service (see the corresponding interfaces in Figure 1).

The platform storage services are utilized by both external TARZAN applications to provide a shared storage and by the core TARZAN components to store the platform runtime data. In the second case, the storage services are utilized for a resource registry of various resources accessed and manipulated by the platform (e.g., topic and partition names for the platform bus, or declarations and definitions of tasks in the platform computation components).

For the distributed filesystem and the distributed and synchronized configuration service, Hadoop Distributed File System (HDFS) from *Apache Hadoop* [2] and *Apache Zookeeper* [1] were adopted, respectively. Both software products are widely utilized in the TARZAN platform and well-integrated with other components. For example, the platform bus based on Apache Kafka is utilizing Zookeeper for message queue management and the platform computation component based on Apache Spark is utilizing HDFS for a data storage and Hadoop for a cluster management.

Although the distributed database service is not designated for a particular NoSQL database, *Apache Cassandra* [3] is the preferred database in the TARZAN platform. The main reason for this preference is a perfect integration of Cassandra with the rest of the software stack (e.g., well-established Cassandra-Spark and Cassandra-Kafka connectors). Moreover, Cassandra provides an optimal storage for large sensor data [23].

4.3 Platform Computation

To support distributed computing on data communicated and stored in the TARZAN platform, the platform computation core component is provided. The component allows TARZAN applications to run tasks, e.g., to process (normalize/aggregate), enrich, label, combine, etc. the data and to utilize other TARZAN components.

Tasks for the platform computation component are registered by external application components and then they can be executed by TARZAN applications (for the corresponding interfaces, see Figure 1) as demonstrated in Figure 3 to perform malware or data-breach detections, or to analyse Bitcoin transaction based on capture network traffic, firewall logs, Bitcoin blockchain, and social network profiles.

As the most of the use-cases for data processing in the TARZAN platform operate on big data (in the sense of data characterized by four Vs: volume,

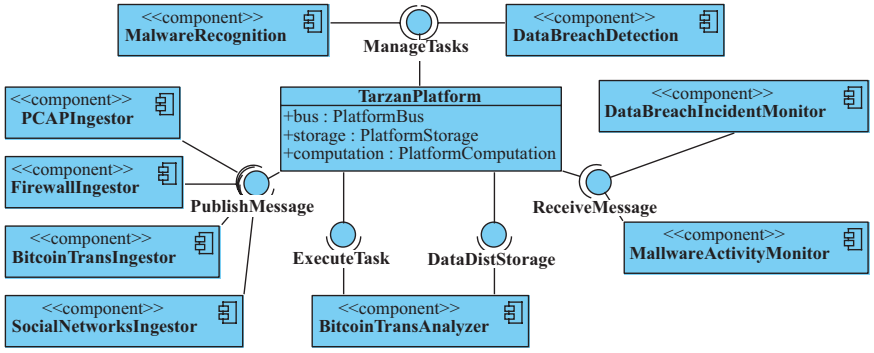


Figure 3 An example of external application components utilizing the TARZAN platform (the ingestors on the left side are feeding data to the platform, computation tasks and an application on the top and bottom are processing the data, and the monitors on right side are passing results to clients).

variety, velocity, and value [12]), the platform computation tasks must be able to do big data processing. The applications need to process both data streams and data batches (e.g., to do a real-time analysis of firewall logs and to execute on-demand tasks, respectively). Therefore, *Apache Spark* [6] has been selected as the implementation technology for the platform computation component and it tasks, as it supports both the stream and batch processing of big data.

For the batch data processing in Spark, computation tasks can utilize a data abstraction called *Resilient Distributed Dataset* (RDD) to execute various parallel operations on a Spark cluster and to gather resulting data in shared broadcast variables and accumulators provided by Spark on the cluster's nodes. In the case of the stream data processing, Spark Streaming provides computation tasks with *Discretized Stream* (DStream) abstraction where each stream is represented by a continuous series of RDDs that is divided into micro-batches and processed by the tasks in the similar way as in the batch processing above. Because DStreams follow the same processing model as batch systems, the two can naturally be combined [25] and the platform computation component and it tasks can implement identical algorithms for both the stream and batch processing.

5 Evaluation

The TARZAN platform has been evaluated by means of the PCAP analysis case study described in Section 3. A corresponding TARZAN application has

been implemented to read and analyse data of network traffic monitoring stored in the PCAP format. After the PCAP data are read from input PCAP files or real-time generated by network traffic monitoring tools, they are transferred (including their related meta-data) via platform bus for a primary analysis by tasks of platform computation. The tasks also ensure that both the input data and the output primary analysis results are stored in platform storage. Afterwards, a secondary analysis can be executed on the stored data and the previous results to perform various security and forensic analyses, e.g., to detect communication patterns, apply clustering methods for data mining, etc.

The primary analysis is operating on continuously incoming data and the primary analysis tasks implement real-time stream processing to quickly extract traffic basic features such as source and destination IP addresses and port numbers, defragment fragmented packets into network flows, analyse flow properties, application protocols, etc. These tasks utilize the Spark Streaming extension of the core Apache Spark API to process DStreams. In Spark, tasks are scalable, high-throughput, fault-tolerant, so the ability to process the incoming live data in real-time can be improved, if necessary, by an appropriate cluster configuration and the application deployment. However, the primary analysis must perform only basic analytical tasks.

Contrary to the primary analysis which employs real-time steam processing, the secondary analysis can implement a batch processing of the previously stored data and the primary analysis results. Therefore, the stored inputs can be represented as RDDs and processed by means of Spark RDD API, Spark SQL, and also machine learning algorithms provided by Spark's machine learning library (MLlib) can be applied. The secondary analysis is executed on demand as required by the platform's client applications, e.g., to provide data for visualisations, analyse network communications related to security incidents under investigation, or related to cryptocurrency transactions or malware activities.

The overall architecture of the PCAP Analysis tool is depicted in Figure 4. To feed input PCAP data into the system, several modules were adopted and adapted from the *Apache Metron* project [5], namely: metron-sensors, metron-pcap, and metron-api. In the first module, Apache Metron brings into TARZAN the integration of *Data Plane Development Kit*[3] (DPDK) probes for high speed packet capture and *Yet Another Flowmeter*[4] (YAF) to processes packet data from PCAP dumpfiles (as generated by tcpdump or libpcap). The next

[3]http://dpdk.org/

[4]https://tools.netsa.cert.org/yaf/

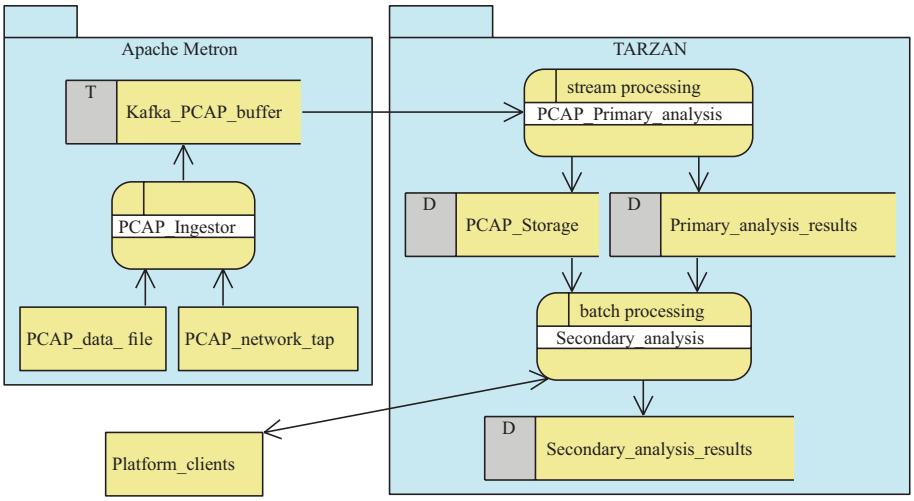


Figure 4 Architecture of the PCAP Analysis tool with data-flows (including processes, data storages, and external data sources and entities).

two Metron modules provide a topology for streaming network packets into HDFS and low-level analytics/filtering on the PCAP files in HDFS before they are submitted into a Kafka message queue acting as a buffer for further processing. Then, a continuous stream processing in the primary analysis and an on-demand batch processing in the secondary analysis is performed by utilizing the TARZAN platform components as described above.

In comparison with the *Apache Metron* [5] or *Apache Spot* [7] discussed in Section 2, the current implementation of the PCAP analysis tool in TARZAN provides the same basic functionality, however, it enables a better integration with the other TARZAN applications into a seamless security analysis framework where results of the PCAP analyses may contribute to various security investigations, e.g., to trace cryptocurrency transactions or malware activity.

In comparison with the existing approaches and the related work (see Section 2), the TARZAN platform is a step further in the design and development of open forensic platform capable of processing big data. As we demonstrated in the PCAP analysis case study, our approach is compatible and easily integrated with other approaches to big data forensic. TARZAN applications can utilize HDFS as suggested in a conceptual model of big data forensics by Zawoad and Hasan [26]. Also a framework for the forensic analysis of big heterogeneous data presented by Mohammed et al. [18] can be

realized using the TARZAN platform. Their framework has three layers that follow acquisition, examination, and analysis approach to extract metadata from acquired data sources and build a semantic web-based model for further analysis. While they do not specify the particular implementation of such system, the presented concepts are in accordance with the architecture of the TARZAN platform. Analogously, Irons and Lallie [15] discussed the shortcomings of the current analysis methods and suggested to use more intelligent techniques and demonstrated the possible application of artificial intelligence (AI) to computer forensics. The TARZAN platform can easily integrate the AI investigative methods because the underlying components provide rich libraries of various AI algorithms.

6 Experimental Setup and Results

Experiments were conducted demonstrating the performance of the proposed TARZAN platform for PCAP analysis. The two main resources controlled by Spark are CPU and memory. Thus the experiments also attempt to provide results for different parameters that control resource utilization.

The dataset used during experiments consists of 10 GB of PCAP files containing about 50 million packets captured in Honeypot network during six months. In the capture, 99,582 unique IP addresses were identified communicating with the Honeypot devices most often by protocols such as SSH, HTTPS, SSDP, DNS, SMB, and NTP. The dataset was split into 102 individual files of average size about 100 MB and uploaded to HDFS.

Supermicro SuperTwin2 6026TT-TF server equipped with eight Intel (R) Xeon E5520 @ 2.26 GHz was the hardware platform utilized in the experiments. This four systems (nodes) server hosts both HDFS and Spark cluster. Spark cluster consists of a master node and four workers. Each node has 16 CPU cores and at least 48 GB RAM.

6.1 Scenarios

Experiments were executed in the spark-shell environment for different settings of memory and core limits for executors[5]. The workload spans from computing the statistics for the entire dataset through enumerating individual flows to analyse application level information. From many possible

[5]Limits were set using “executor-memory” and “executor-cores” options.

```
val frames = sc.hadoopFile("hdfs://storage/cap/*.cap", ...)
val packets = frames.map(x => Packet.parsePacket(
        x._2.get().asInstanceOf[RawFrame]))
val capinfo = packets.map(Statistics.fromPacket)
        .reduce(Statistics.merge)
```

(a) Capture Info

```
val frames = sc.hadoopFile("hdfs://storage/cap/*.cap", ...);
val packets = frames.map(x => Packet.parsePacket(
        x._2.get().asInstanceOf[RawFrame]))
val flows = packets.map(x =>(x.getFlowString(),x))
val stats = flows.map(x => (x._1, Statistics.fromPacket(x._2)))
        .reduceByKey(Statistics.merge)
stats.takeOrdered(10)(
        Ordering[Int].reverse.on(x=> x._2.getPackets()))
        .map(c=>Statistics.format(c._1, c._2)).foreach(println)
```

(b) Flow Aggregation

```
def xf(x:PacketPayload)= x.getPacket().extendWith("http",
        HttpRequest.tryParseRequest(x.getPayload()))
val packets = frames.map(x => Packet.parsePacket(
        x._2.get().asInstanceOf[RawFrame], toConsumer(xf)))
val http = packets.filter(x => x.containsKey("http.request"))
val hosts = http.map(x => (x.get("http.host"),1))
        .reduceByKey(_ + _)
hosts.takeOrdered(20)(Ordering[Integer].reverse.on(x => x._2))
        .foreach(println)
```

(c) HTTP Request Analysis

Figure 5 Scala code snippets for test scenarios.

operations, the following three were selected representing different types of PCAP processing:

- Capture Info. It computes basic information about the packet capture collection similarly to `capinfo` tool from Wireshark distribution. It provides information such as the number of packets, capture duration, data byte rate, average packet size, average packet rate, etc. The computation is done in a simple one stage pipeline that reduces to a single object: The code is shown in Figure 5(a).
- Flow aggregation. The packets are aggregated into flows, and for each flow, the usual statistical information is computed. For instance, the top

10 flows with the most packets are computed as follows: The code is shown in Figure 5(b).

- HTTP Request Analysis. This scenario is similar to CloudShark's HTTP analysis tool[6]. For instance, HTTP requests by hosts can be enumerated by the following sequence: The code is shown in Figure 5(c).

The presented cases represent the different computations performed by the TARZAN platform during processing and querying packet capture data source. Computation time was measured to estimate the performance. All three cases were tested with different parameters limiting the number of CPU cores and memory for executors. Spark adjusts the number of executors to available resources. For our hardware configuration, Spark allocates executors as shown in Figure 6(f) for different combinations of executor's memory size and cores.

6.2 Results

Performance measured for the first case is shown in Figure 6(a). The worst time is for a configuration that instantiates only four executors each allocated 16 GB memory and assigned 16 cores. The best computation time was achieved by the configuration creating 32 executors each assigned only two cores and 4 GB of memory. Configurations that allocates 4 CPU cores per executor and more are not better – they have almost equal running time.

A possible explanation may be that HDFS I/O is the limiting factor in this situation.

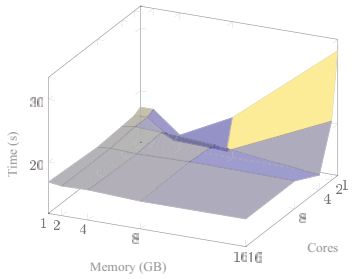
Figure 6(b) shows measured performance for the second case. Also, in this case, the worst performance is achieved for large memory allocation and a small number of executors. Contrary to the previous situation, using 8 CPUs per executor leads to the best performance. Using more CPUs or more memory does not improve the performance.

The results for the third cases are given in Figure 6(c). Results for HTTP Request Analysis are close to Flow Aggregation case except for tiny executors that exhibit poor performance. The best computation time is about 20 seconds.

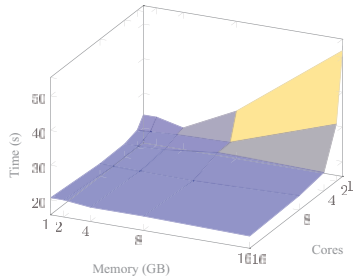
For comparison, Figure 6(d) shows computation time for the program that reads all frames from packet capture files and performs count operation on the RDD. The best computation time (12 seconds) was achieved for 17 executors each with only 1CPU and 8 GB.

Figure 6(e) provides results on the impact of the total number of executors on the job completion time. An executor runs tasks both sequentially and in parallel. The general advice is to have at least as many executors as data

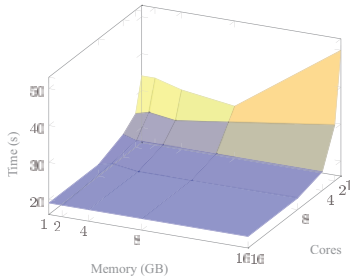
[6]see https://supports.cloudshark.org/user-guide/analyze-http-requests.html



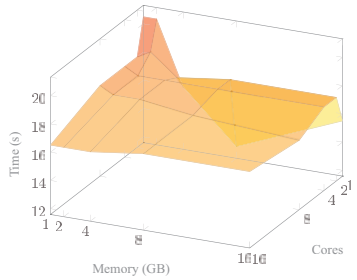
(a) Computation time for Capture Info



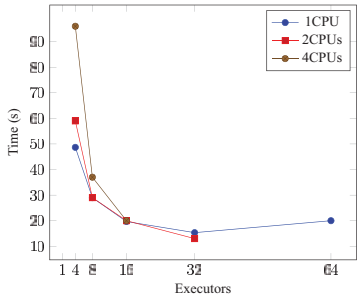
(b) Computation time for Flow Aggregation



(c) Computation time for Http Request Analysis



(d) Computation time for count() operation



(e) Capture Info : Computation time for different size of executors

Mem (GiB)	vCPU 1	2	4	8	16
1	64	32	16	8	4
2	59	32	16	8	4
4	38	29	16	8	4
8	17	17	14	7	4
16	7	7	7	7	4

(f) Number of executors created by Spark for different configuration of memory and CPU

Figure 6 Computation time for experimental scenarios.

nodes. It is possible to create as many executors as there are cores in the cluster. However, executors with a single core and small amount of memory will not be able to run tasks in parallel that may lead to additional communication and data duplication overhead. Also, executors with too much memory often result in excessive garbage collection delays. In this scenario, each executor has allocated 2GB of memory. The best computation time was achieved by 32 executors each having assigned two CPUs. Using 4CPUs per executor instead of two did not improve the performance. The configuration that consists of 4 executors each with 4CPUs had the worst performance.

6.3 Discussion

The presented performance results necessarily reflect the hardware platform utilized and the cluster organization. However, some useful observations can be made:

- Tiny executors perform better for simple workloads. Moreover, many small executors beat heavy executors for the same available hardware resources.
- Allocating more memory for executors usually does not lead to better performance.
- Four to eight cores per executor is a good option for the considered workload.
- In all scenarios, there is a configuration space that provides reasonable performance, which simplifies system parameter tuning.

As seen, the performance is adequate for real scenarios. For comparison, obtaining capture information with `capinfo` tool form Wireshark distribution takes about 3 minutes for the considered dataset. The presented results are for proof of concept implementation that does not involve any optimization. The amount of parallelism can be further tuned for instance by creating smaller capture files or modifying InputFormat to create more splits. Another option is to optimize data structures used for representing packets, flows and other data structures and their serialization format.

7 Conclusion

In this paper, we have introduced a TRAZAN platform, an integrated platform for analysis of digital data from security incidents. The architectural design has been presented to explain which core component are available in

the platform and which services are provided to TARZAN applications. The platform allows to gather, store, and process digital forensic data as big data to perform various security-oriented analyses.

As a sample case study, a PCAP analysis tool has been implemented on the platform utilizing the platform bus component to integrate individual modules, the platform storage component to store input data and analyses results, and the platform computation component to perform both stream and batch processing of big data.

It has been concluded that the TARZAN platform constitutes an open forensic platform capable of processing big data and provides a sufficient framework for further integration of various existing approaches. The integration of various existing approaches and existing tools for forensic analyses as external TARZAN components and applications is a part of ongoing work.

Acknowledgements

This work was supported by Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science" LQ1602; Ministry of Interior of the Czech Republic project "Integrated platform for analysis of digital data from security incidents" VI20172020062; and by BUT internal project "ICT tools, methods and technologies for smart cities" FIT-S-17-3964.

References

[1] Apache ZooKeeper, 2010.
[2] Welcome to Apache Hadoop! 2014.
[3] Apache Cassandra, 2016.
[4] Apache Kafka: A high-throughput distributed messaging system, 2016.
[5] Apache Metron: Real-time big data security, 2016.
[6] Apache Spark: Lightning-fast cluster computing, 2016.
[7] Apache Spot (incubating). (2016). A community approach to fighting cyber threats.
[8] Aupetit, M., Zhauniarovich, Y., Vasiliadis, G., Dacier, M., and Boshmaf, Y. (2016). Visualization of actionable knowledge to mitigate DRDoS attacks. In *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*, (pp. 1–8). IEEE.

[9] Cardenas, A. A., Manadhata, P. K., and Rajan, S. P. (2013). Big data analytics for security. *IEEE Security and Privacy*, 11(6), 74–76.

[10] Cohen, M. I. (2008). Pyfiag: An advanced network forensic framework. In *Proceedings of the 2008 Digital Forensics Research Workshop*. DFRWS.

[11] Casey, E. (2004). Network traffic as a source of evidence: tool strengths, weaknesses, and future needs. *Digital Investigation*, 1(1), 28–43.

[12] Gantz, J., and Reinsel, D. (2011). Extracting value from chaos. *IDC iview*, 1142(2011), 1–12.

[13] Guarino, A. (2013). Digital forensics as a big data challenge. In *ISSE 2013 securing electronic business processes* (pp. 197–203). Springer Vieweg, Wiesbaden.

[14] He, L., Tang, B., Zhu, M., Lu, B., and Huang, W. (2016). NetflowVis: A Temporal Visualization System for Netflow Logs Analysis. In *International Conference on Cooperative Design, Visualization and Engineering* (pp. 202–209). Springer, Cham.

[15] Irons, A., and Lallie, H. S. (2014). Digital forensics to intelligent forensics. *Future Internet*, 6(3), 584–596.

[16] Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., and Shahabi, C. (2014). Big data and its technical challenges. *Communications of the ACM*, 57(7), 86–94.

[17] Lukashin, A., Laboshin, L., Zaborovsky, V., and Mulukha, V. (2014). Distributed packet trace processing method for information security analysis. In *International Conference on Next Generation Wired/Wireless Networking* (pp. 535–543). Springer, Cham.

[18] Mohammed, H., Clarke, N., and Li, F. (2016). An automated approach for digital forensic analysis of heterogeneous big data. *JDFSL*, 11(2), 137–152.

[19] Pilli, E. S., Joshi, R. C., and Niyogi, R. (2010). Network forensic frameworks: Survey and research challenges. *Digital Investigation*, 7(1–2), 14–27.

[20] Promrit, N., and Mingkhwan, A. (2015). Traffic flow classification and visualization for network forensic analysis. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications (AINA)*, (pp. 358–364). IEEE.

[21] Quick, D., and Choo, K. K. R. (2016). Big forensic data reduction: digital forensic images and electronic evidence. *Cluster Computing*, 19(2), 723–740.

[22] Schales, D. L., Hu, X., Jang, J., Sailer, R., Stoecklin, M. P., and Wang, T. (2015). FCCE: highly scalable distributed feature collection

and correlation engine for low latency big data analytics. In *2015 IEEE 31st International Conference on Data Engineering (ICDE)*, (pp. 1316–1327). IEEE.

[23] Van der Veen, J. S., Van der Waaij, B., and Meijer, R. J. (2012). Sensor data storage performance: SQL or NoSQL, physical or virtual. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, (pp. 431–438). IEEE.

[24] Wullink, M., Moura, G. C., Müller, M., and Hesselman, C. (2016). ENTRADA: A high-performance network traffic data streaming warehouse. In *2016 IEEE/IFIP Network Operations and Management Symposium (NOMS)*, (pp. 913–918). IEEE.

[25] Zaharia, M., Das, T., Li, H., Shenker, S., and Stoica, I. (2012). Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'12, Berkeley, CA, USA, 2012. USENIX Association.

[26] Zawoad, S., and Hasan, R. (2015). Digital forensics in the age of big data: Challenges, approaches, and opportunities. In *2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conference on Embedded Software and Systems (ICESS)*, (pp. 1320–1325). IEEE.

Biographies

Marek Rychlý is an assistant professor at Brno University of Technology, Faculty of Information Technology (BUT FIT). He received PhD in Computer Science and Engineering in 2010 from BUT FIT. His research interests are in the area of software architecture and focus on dynamic reconfiguration and

component mobility in component-based and service-oriented architectures, formal description of software architectures and their evolution, functional and quality-driven automatic Web services composition and testing, and on distributed software systems. He authored over 20 scholarly journal articles and conference papers on varied topics related to software engineering and software architectures.

Ondřej Ryšavý is an associate professor at Brno University of Technology (Czech Republic). He has a PhD in Information Technology. His research projects include Programmability in Rina for European Supremacy of virtualised Networks, Modern Tools for Detection and Mitigation of Cyber Criminality on the New Generation Internet, SCADA system for control and monitoring RT processes and Dependable Systems International Research and Educational Experience.