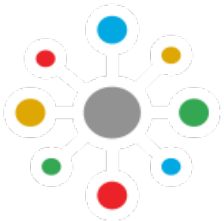
TARZAN: An Integrated Platform for Security Analysis

Marek Rychlý & Ondřej Ryšavý

Brno University of Technology
Faculty of Information Technology
Department of Information Systems
(Czech Republic)

Federated Conference on Computer Science and Information Systems (FedCSIS'17)
3 – 6 September, 2017

The 4th International Conference on Cryptography and Security Systems (C&SS'17)



Outline

IT Forensic (Security) Analysis

- Organisations need to collect, classify, and analyse forensic evidences and data in IT.
  (law enforcement agencies, IT security departments, auditors, etc.)
- With increasing network bandwidth, storage size, mobile devices propagation, etc., there is a lot of IT forensic data to process.
- The forensic data needs to be processed as Big data.
  (large volume, high velocity and variety, questionable veracity, unknown value)
- Common/legacy analytical approaches are not usable any more.
- Distributed, highly scalable and flexible processing is needed.

Current Problems in Related Work

- There are several (mostly proprietary) tools to collect and process IT forensic evidence in various domains.
  (network traffic, personal/mobile computer devices, instant messaging and over-IP calls, e-commerce transactions, cryptocurrencies, social network communication and relationships, etc.)
- These tools specialise in the individual domains and it is difficult **to integrate them or to scale** them up for Big data.
- The tools process forensic data in batches, **not real-time data**.
  (PCAP files, storage/memory dumps, physical devices, etc.)
- Yet, a forensic investigation typically needs to provide facts that confirm/deny something for criminal and judicial proceedings – it requires the ability **to put together** various forensic information.
  (e.g., to prove/deny that a suspect performed a particular action or a sequence of actions in particular location on particular objects by particular means with particular knowledge)

TARZAN Project: Goals

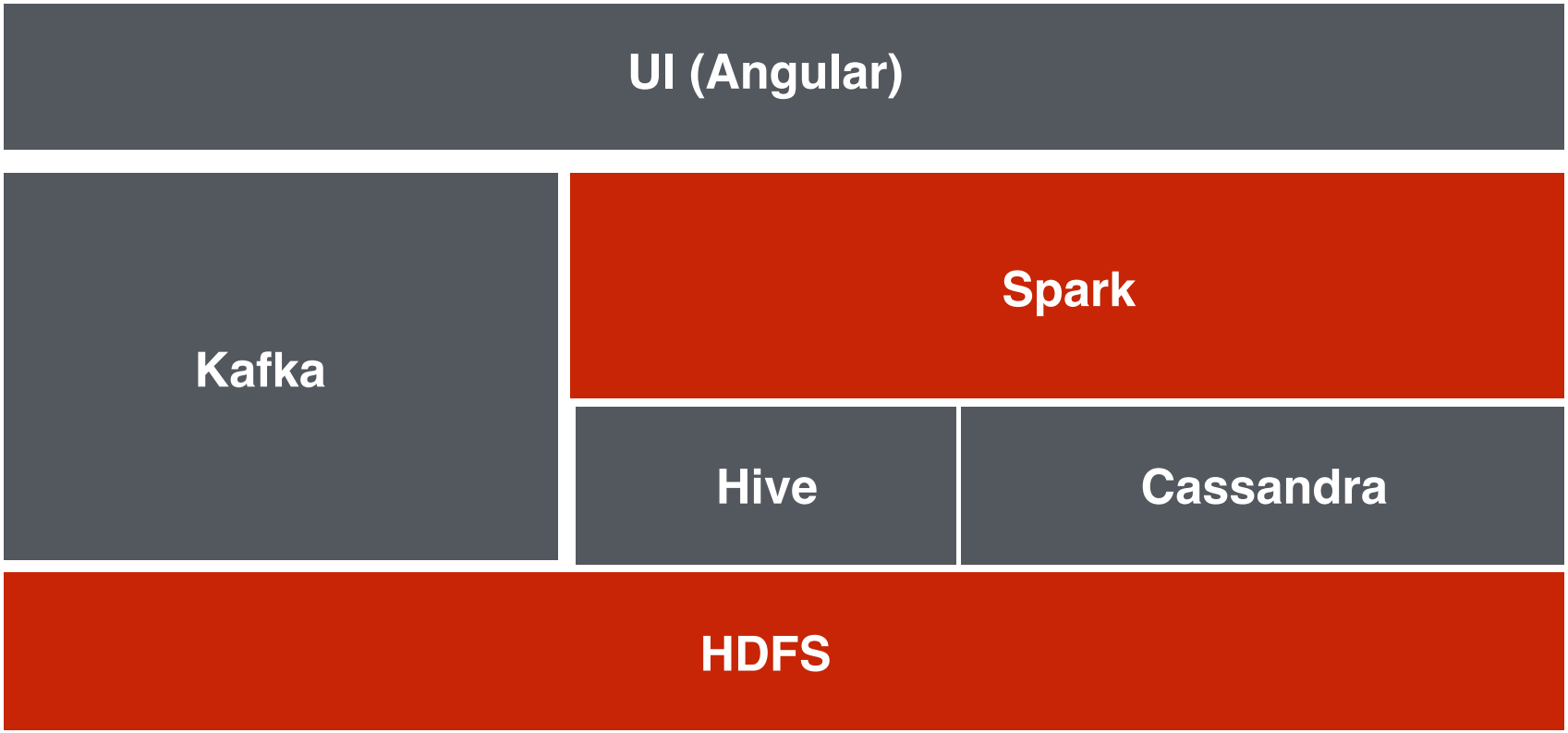
An Integrated Platform for Security Analysis

- To support innovative ways of investigative analysis in IT.
  (e.g., to analyse coincidences and causations in the forensic information)
- To provide scalability for large forensic data-sets & real-time data.
  (we are processing Big data)
- To allow an open integration of various forensic analysis tools.
  (both new tools, e.g., for crypto-currency analyses, and legacy tools, e.g., for mobile phone device analyses)
- To provide rapid forensic application development platform.
  (with the ability to easily integrate predefined components through well-established interfaces, to build flexible or ad-hoc analyses on demand)

TARZAN Architectural Components

- Software/Tools for TARZAN applications
  - to integrate data/outcomes of individual forensic tools
  - to provide platform services to the tools / research groups
- The three core components:
  1. Platform Bus – asynchronous communication of components
     (message oriented middleware/MOM, or message queue/MQ)
  2. Platform Database(es) – a shared data storage & resource registry
     (Big data/Fast data: PCAPs, images/dumps, table-rows, etc.)
  3. Platform Computation – running distributed computing tasks
     (mostly, however, not only, reactive processes on messages/data)
- A set of utility services to access/control the components.
  (a predefined set of services to publish/subscribe, access data & tasks)

TARZAN Architecture – Technologies



Platform Bus

- To enable asynchronous communication of components.
  (message oriented middleware/MOM, or message queue/MQ)
- Publisher-subscriber/PS model with load-balancing (scalability).
  (producers send their data to defined topics/MQs, consumers read the data)
- High throughput and support for large data.
  (must be simple – no VETRO pattern[1])
- Technology: Apache Kafka

  message a key-value pair to be transmitted in a topic
  (from a producer to one or more consumers/partitions)

  topic a general category of messages
  (just a few of topics, representing main services)

  partition to split a particular topic on partitions/queues
  (each stored on a single machine, read by a single consumer)

[1] validate, enrich, transform, route, operate

Platform Database(s)

- Distributed NoSQL database(es).
  (it is necessary for distributed computing on the hosted data)
- **Platform Registry** of addresses of shared resources.
  (both internal and external resources; by uuid, URI, MQ labels, etc.)
- Technology: Cassandra
  - a key-value distributed database (SQL-like NoSQL)
    (a hash-map by row-key of ordered linked-hash-map of column-key values)
  - suitable for large collections of raw data (e.g., metrics)
    (however, also for any key-value structured data of a particular schema)
- Another NoSQL databases are supported as well.
  (on on-demand basis, together with corresponding services)
- **Platform File Storage** supported by HDFS
  (a distributed FS with POSIX interface for sequential access)
- **Platform Sync. Service** for distributed configs/handshake.
  (required by MQ/Kafka in distributed environments; by Apache ZooKeeper)

Platform Computation

- To run distributed computing tasks.
  (mostly, however, not only, reactive processes on messages/data)
- To implement consumers and producers on MQs.
  (e.g., to read a raw data, perform an analysis, and to store/publish results)
- Apache Hadoop as a standard for scalable distributed computing.
  (it serves as an eco-system for other big-data processing tools)
- Technology: Apache Spark
  - easy to use (a declarative approach)
    (“apply a given algorithm on data”, not “read/write data in a particular way”)
  - for both batch/stream data processing
    (for the publish/subscribe model of Platform Bus)
  - can be combined with advanced processing models
    (e.g., an actor model by Akka)

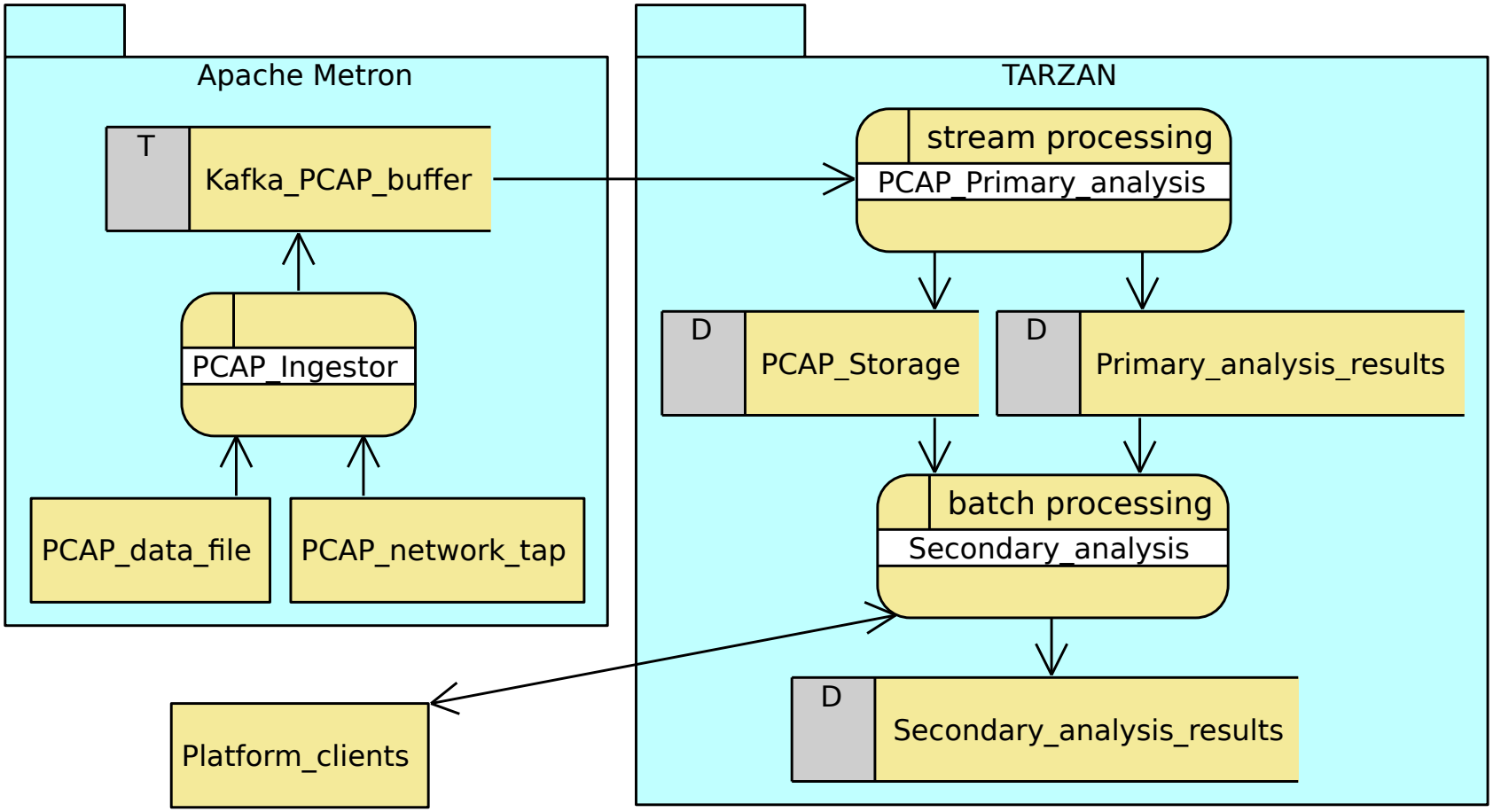
TARZAN Ingestors and Client Applications

- network packet (PCAP/NetFlow) analysis
  (statistics with a particular focus, application specific information, etc.)
- social network analyses
  (conversations, contact lists, profiles, etc.)
- distributed denial-of-service attack investigations
  (the ability to put together network loads from various malicious sources)
- internet-of-things monitoring and malicious activity detection
  (identification of various attack vectors, to individual nodes, network, protocols, application, etc.)
- crypto-currency transactions analyses
  (block-chain analyses, identification of transaction parties, etc.)

Experiment: PCAP Analyser

- to ingest and analyse packet network traffic (PCAP data)
  (the high-level analysis of application-specific protocols: HTTP, SMB, SMTP, . . . ; statistics, anomaly/keyword detection, content extraction, identification, etc.)
- distributed processing of very large PCAP data-sets (GBs)
  (it is impossible to efficiently perform such analyses with current tools, e.g., Wireshark, Network Monitor, Tshark, editcap, capinfo, etc.)
- three components utilizing the TARZAN platform:
  (Kafka, Cassandra, Spark)
  - a data-source adapter for ingestion
    (loading, pre-processing, and sending into the platform)
  - a worker for analytics
    (receiving from the platform, analysing, and storing results in the platform)
  - an application for UI
- Integration of Apache Metron for the ingestion.

Dataflow in the PCAP Analyzer



Flexibility and Scalability

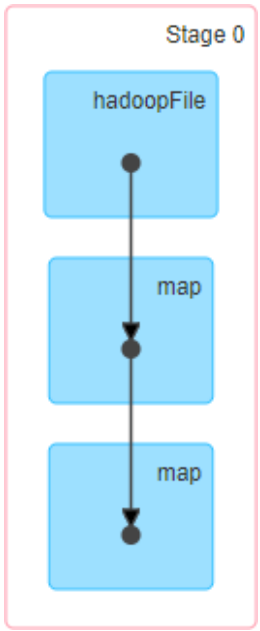
- The components can be distributed across a cluster.
  (they are asynchronously communicating via Kafka message queues)
- There are typically multiple data-source adapters.
  (both for network real-time monitoring and for accessing stored PCAP datasets)
- The workers performing the analyses can be parallelised.
  (they are Spark nodes in a Hadoop cluster)
- Both predefined and custom ad-hoc analyses can be performed.
  - capinfo – to investigate captured PCAP data
  - flowstats – to investigate data-flows in the captured PCAP data
  - xinfo – to investigate application-specific protocols in the data
- The analyses can be integrated into other TARZAN applications.
  (e.g., they can help to identify parties in crypto-currency transactions)

PCAP Analyser: Experimental Data and Environment

- Hardware:
  SuperTwin2 6026TT-TF, SC827T-R1200B, X8DTT-F, 8xE5520 (2,26GHz Nehalem), 100 GB RAM, 4TB HDD
- Sample data:

```
File name:              192.168.186.18.cap
File type:              Wireshark/tcpdump/... - pcap
File encapsulation:     Ethernet
File timestamp precision:  microseconds (6)
Packet size limit:      file hdr: 65535 bytes
Number of packets:      50 M
File size:              11 GB
Data size:              10 GB
Capture duration:       24789795.090000 seco
First packet time:      2015-04-20 13:49:56.140000
Last packet time:       2016-02-01 10:53:11.230000
Data byte rate:         417 bytes/s
Data bit rate:          3337 bits/s
Average packet size:    203.39 bytes
Average packet rate:    2 packets/s
SHA1:                   97efe62aaa42402d3b84292d1fc010a5090f3332
RIPEMD160:              e711effc6f6c47490f886df5fd7941cfc5df1b47
MD5:                    54844e4f6d9fa58f30b293d64a6b4150
```

PCAP “capinfo” Analysis

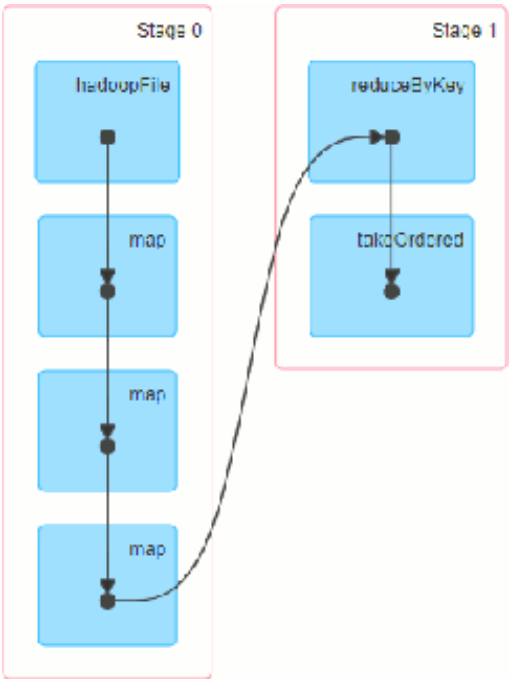



```
import org.ndx.model.Packet;
import org.ndx.model.PacketModel.RawFrame;
import org.ndx.model.Statistics;

val frames = sc.hadoopFile("hdfs://neshpc1.fit.vutbr.cz/user/rysavy/cap/*.cap",
classOf[org.ndx.spark.pcap.PcapInputFormat],
classOf[org.apache.hadoop.io.LongWritable],
classOf[org.apache.hadoop.io.ObjectWritable])

val packets = frames.map(x=> Packet.parsePacket(x._2.get().asInstanceOf[RawFrame]))

val capinfo = packets.map(x => Statistics.fromPacket(x)).reduce(Statistics.merge)
```

Processing: 12 seconds (102 tasks)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input
0	reduce at <console>:31 +details	2017/08/25 18:00:39	12 s	102/102	10.4 GB

PCAP “flowstat” Analysis



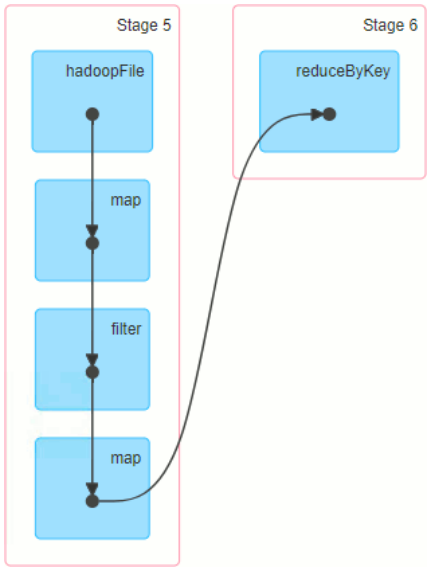

```
import org.ndx.model.Packet;
import org.ndx.model.PacketModel.RawFrame;
import org.ndx.model.Conversations;

val frames = sc.hadoopFile("hdfs://meshpc1.fit.vutbr.cz/user/rysavy/cap/*.cap",
classOf[org.ndx.spark.pcap.PcapInputFormat],
classOf[org.apache.hadoop.io.LongWritable],
classOf[org.apache.hadoop.io.ObjectWritable]);
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,Conversations.fromPacket(x._2))).reduceByKey(Conversations.merge);
println("Date flow start          Duration Proto   Src IP Addr:Port      Dst IP
Addr:Port     Packets     Bytes Flows")
stats.takeOrdered(10)(Ordering[Int].reverse.on(x=>
x._2.getPackets())).map(c=>Conversations.format(c._1, c._2)).foreach(println)
println("Done.")
```

Processing: 3 + 17 seconds (102 tasks in each stage)

Completed Stages (2)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	takeOrdered at <console>:36	+details	2017/08/23 14:45:06	3 s	102/102			191.5 MB	
0	map at <console>:33	+details	2017/08/23 14:44:48	17 s	102/102	10.4 GB			191.5 MB

PCAP “xinfo” Analysis




```
implicit def toConsumer[A](function: A => Unit): java.util.function.Consumer[A] = new java.util.function.Consumer[A]() {
  override def accept(arg: A): Unit = function.apply(arg)
}


import org.ndx.model.Packet;
import org.ndx.model.PacketPayload;
import org.ndx.model.PacketModel.RawFrame;
import org.ndx.model.Statistics;
import org.ndx.tshark.HttpRequest;

val frames = sc.hadoopFile("hdfs://nosh-pc1.fit.vutbr.cz/user/rysavy/cap/*.cap",
                           classOf[org.ndx.pcap.PcapInputFormat],
                           classOf[org.apache.hadoop.io.LongWritable],
                           classOf[org.apache.hadoop.io.ObjectWritable])
```

Processing: 0.05 + 30 seconds (1 + 102 tasks)

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
6	take at <console>:42	+details	2017/09/02 14:05:14	50 ms	1/1			26.7 KB	
5	map at <console>:39	+details	2017/09/02 14:04:44	30 s	102/102	10.4 GB			2.6 MB

Summary and Future Work

- The platform to gather, store, and process digital forensic Big data.
  (various security-oriented analyses, both predefined and custom/ad-hoc)
- PCAP Analyser utilizes
  - the platform bus component to integrate individual modules
  - the platform storage component to store analyses results
  - the platform computation component for stream/batch processing.
- The open forensic platform capable of processing Big data.
- Sufficient for further integration of various existing approaches.

Future work

- more experiments
  (especially with various data-source adapters and deployment configurations)
- better integration with other tools
  (both in the TARZAN project and external tools for forensic analyses)

Thank you for your attention!

Marek Rychlý

<rychly@fit.vutbr.cz>

This work was supported by:

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