A Big Data Architecture for Large Scale Security Monitoring

Samuel Marchal*†, Xiuyan Jiang‡, Radu State*, Thomas Engel*

* SnT - University of Luxembourg, Luxembourg – { samuel.marchal | radu.state | thomas.engel }@uni.lu
† TELECOM Nancy - University of Lorraine, France – samuel.marchal@univ-lorraine.fr
‡ Faculty of Sciences, Technology and Communication - University of Luxembourg, Luxembourg – kangsyeun@gmail.com

***Abstract*—Network traffic is a rich source of information for security monitoring. However the increasing volume of data to treat raises issues, rendering holistic analysis of network traffic difficult. In this paper we propose a solution to cope with the tremendous amount of data to analyse for security monitoring perspectives. We introduce an architecture dedicated to security monitoring of local enterprise networks. The application domain of such a system is mainly network intrusion detection and prevention, but can be used as well for forensic analysis. This architecture integrates two systems, one dedicated to scalable distributed data storage and management and the other dedicated to data exploitation. DNS data, NetFlow records, HTTP traffic and honeypot data are mined and correlated in a distributed system that leverages state of the art big data solution. Data correlation schemes are proposed and their performance are evaluated against several well-known big data framework including Hadoop and Spark.**

I. Introduction

The detection and prevention of network intrusions is a recurrent security problem. It has been studied for over thirty years [1] with the first concept of Intrusion Detection System (IDS) being proposed in 1987 [2]. However it remains an open research topic due to the constant evolution of types of data to analyse. In addition, the adaptation of attackers' techniques to cope with new means of protection and firewall policy makes it a continuously evolving field. Moreover, it raises new challenging issues related to identifying relevant features for intrusion detection, as well the means of processing the increasing volume of heterogeneous security data produced by a network.

The operations of Network Intrusion Detection Systems (NIDS) rely on network traffic analysis, where Snort [3], Bro [4] and Suricata [5] are typical examples. Network traffic from several protocols (HTTP, SIP, DNS, etc.) is inspected to find anomalies. These anomalies are defined by rules that rely on either signatures or anomalous traffic behaviour. If such anomalies are observed, the system either raises an alert (IDS) or stops the communication (IPS). Current IDSs analyse several protocols and data and events observed by them are correlated by SIEM (Security Information and Event Management) in order to detect intrusions. One shortcoming is that current solutions realizing in-depth packet analysis are not scalable and adaptable to big network producing high quantity of data.

Operators, *i.e.* the Internet Service Providers (ISPs), have to deal with huge quantities of traffic data. Due to such scalability issues, ISPs usually collect IP flow data as this represents an aggregated view of traffic by discarding the payload. This also preserves the privacy of end users. Main IP flow record attributes are source and destination IP addresses, source and destination ports, the version of IP protocol, a timestamp, the number of bytes and number of packets exchanged. However, not only the volume of data and its velocity but also the variety of information is a challenge. Data can come from different sources: honeypots, DNS monitoring, standalone intrusion detection systems, such that having an unified approach for its analysis is needed.

We have addressed this issue in this paper and proposed a system architecture dedicated to intrusion detection and prevention of a local company network. The proposed approach relies on the inspection of several relevant data sources such as DNS traffic, HTTP traffic, IP flow records and honeypot data. Each of them was already used for intrusion detection individually and proved efficient [6], [7], [8], [9], [10]. However to cope with new hybrid attack techniques that exploit all means and flaws of the network, protection systems must use all data sources available. The proposed system integrates three different data storage systems in the same distributed data storage and processing facility. This data is exploited by a distributed data correlation system to provide a large scale security monitoring system.

The contribution of this paper are:

- we introduce a new intrusion detection architecture that correlates several data sources (HTTP, DNS, IP flow, etc.),
- we propose a solution for processing and storing data coming from different data storage system in a single facility,
- we present data correlation schemes useful for security monitoring and evaluate these against several state of the art distributed computing system including Hadoop and Spark.

The remaining of the paper is structured as follows. In Section II we introduce the architecture of the system and the data correlation schemes as well as their usage. In Section III we present the global distributed data storage and processing facility. We evaluate the performance of correlation operations against several big data frameworks in Section IV. We briefly talk about the related work in Section V and we conclude in section VI.

II. System Architecture and Applications

The global architecture of the proposed security monitoring system is depicted in Figure 1. It is composed of an heterogeneous distributed data storage system — further described in section III — and a distributed data correlation system.

Four kinds of data are gathered to be correlated by our system for security monitoring:

- DNS replies
- HTTP packets
- IP flow records
- Honeypot data

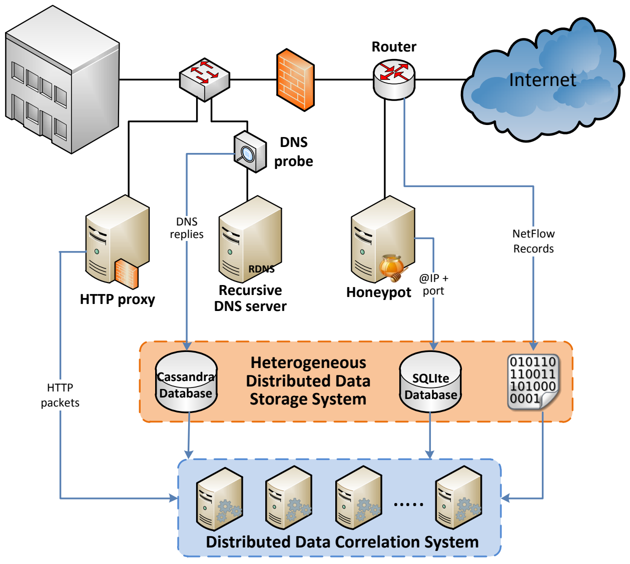

Fig. 1. Large scale security monitoring architecture

A. Data Presentation

Almost all network communications nowadays are initiated via DNS, which is a core service of the Internet. DNS requests are performed to get IP addresses associated with a domain and then consult the needed resource. Hence, monitoring DNS to identify malicious domains is an efficient way to proactively detect and prevent an important part of malicious communications. By setting up a DNS probe at the recursive DNS server of a company network as proposed by Weimer [11], all relevant DNS requests and replies of the network are gathered without data redundancy. The DNS probe of our system is set up according to ISC passive DNS architecture [12]. DNS replies addressed to the recursive DNS server are captured and the information is extracted from DNS packets. Every observed domain along with its extracted information are stored in an Apache Cassandra database [13], a highly scalable and available database.

HTTP traffic represents a significant portion of Internet users' traffic and is a well-known intrusion vector. The study of URIs embedded in HTTP packets and their payload is useful in the detection and prevention of malicious communications. Studying URIs involved in communications between one single machine of the enterprise network and other machines from the Internet is interesting for security monitoring perspectives. Hence, HTTP traffic is monitored at the HTTP proxy level of the network. All HTTP connection attempts of the network go through this proxy and different URIs successively involved in the same communications are analysed on the fly by examining embedded domains and IP destination of the packets. We limit in our system the study to payload and URI embedded in HTTP traffic but the same technique can be applied to any king of protocol including IP addresses and URI such as SIP packets for instance.

Cisco NetFlow [14] is a well-known IP flow monitoring format to observing traffic through routers. This records features about communications forwarded through a router. It consists in source and destination IP addresses, source and destination ports, a timestamp, the protocol used, the amount of data exchanged, etc. Flow records are valuable data to detect intrusions [8] or to highlight botnet communications [9] for instance. By exporting NetFlow records from the core router of the network, we store traces of every communication from the enterprise network to the Internet and vis versa. NetFlow records are distributively stored in `nfcapd` binary files.

The last kind of data used in our system is honeypot data. A honeypot is a machine that seems to be part of a production network but is actually not. It generally emulates vulnerable services and contains fake production data. It is intended to be attacked because it looks like a machine presenting flaws. Connection attempts to a honeypot are proofs of undesired activities, being either the result of misconfiguration, or of probing attacks, with the latter representing the main part. Hence logging honeypot information gives knowledge about attackers targeting a specific network, such as IP addresses used, protocols used, exploit file used, scanning strategies, etc. Exploiting honeypot data allows to adapt security policies in order to prevent attacks perpetrated against production machines by the same attacker. The honeypot solution chosen in our architecture is Dionaea [15], a low interaction honeypot that emulates several vulnerable network services (HTTP, FTP, MSSQL, SMB, etc.). All connections attempts are stored in a SQLite database according to basic implementation of Dionaea.

B. Data correlation

We describe in this section the rationale and processing of the data in the distributed data correlation system. Data from the four sources are combined in order to compute scores depicting the level of risk for communications.

When a domain is queried by a client, the request goes to the local Recursive DNS server. It treats the request by consulting its cash or making recursive requests to authoritative DNS servers to resolve the domain. All the authoritative DNS replies coming to the recursive DNS server are continuously stored in the Cassandra database. As DNS replies are gathered, three operations are performed to infer the maliciousness of a domain:

- Domains are checked against three publicly available blacklists[1][2][3]. For every blacklist the domain belongs to, a score, BL_{dom} is incremented.
- Automated classification techniques relying on DNS data such as [6], [7], [16] are used to identify malicious domains. The confidence score given by the machine learning algorithm is used as a risk score C_{dom}. The closer it is to 1, the more the domain is likely to be malicious.
- Every IP address appearing in DNS resource records are checked against IP addresses logged by the honeypot. For each match a counter H_{dom} is incremented. Assuming a domain having five A resource records *i.e.* five IP addresses associated, if three of these were logged by the honeypot, its score is 3.

These three scores are used to compute a score that quantifies a domain maliciousness in Equation 1. This score, Dom_{score}, is computed and stored along with every domain name in the database.

$$Dom_{score} = \alpha * BL_{dom} + \beta * C_{dom} + \gamma * H_{dom} \quad (1)$$

For every flow exported by the core router three correlation schemes are applied to infer the maliciousness of a communication. These correlation schemes are applied to IP addresses that do not belong to the enterprise network range:

- IP addresses related to a flow are checked against IP addresses logged by the honeypot. If there is a match, ports are checked as well. H_{flow} is computed by adding two boolean values corresponding to IP address matching and port matching. The latter being only computed if there is an IP address matching first.
- IP addresses related to a flow are checked against the DNS database as well to determine if these are associated with domains we consider as malicious. Typically we compute MD_{flow} as a score corresponding to a sum of Dom_{score} for all domains associated with the IP address of a flow record.
- If an IP address appearing in flow records is not associated with any domains of the DNS database, this address has a score $NO_{flow} = 1$. This denotes communications using hard-coded IP addresses, this can be suspicious as they may try to bypass DNS based intrusion detection.

Using these three scores we compute a score determining the risk of a communication described by a flow record. The formula to compute this $Flow_{score}$ is given in Equation 2.

$$Flow_{score} = \alpha * H_{flow} + \beta * MD_{flow} + \gamma * NO_{flow} \quad (2)$$

[1]www.malwaredomainlist.com
[2]www.malwaredomains.com
[3]zeustracker.abuse.ch

HTTP traffic is mined on line and no trace is kept of regular traffic. While HTTP packets go through the proxy they are forwarded to our analysis system to be rated:

- The URI of HTTP packets is checked to determine if it embeds a domain present in the DNS database. Then $D_{web} = Dom_{score}$ is computed to highlight if the URI is malicious.
- HTTP packets having an IP address logged by the honeypot as destination or source IP have a H_{web} score equal to 1 and 0 else. This denotes communication with potentially malicious sources because logged in the honeypot.
- HTTP packets including payload received by the emulated HTTP server of the honeypot have PLh_{web} score equal to 1 and 0 else. This can disclose trace of injecting code or shell scripts in HTTP packets.
- Finally payload of HTTP packets is mined to observe if it includes some domains present in the passive DNS database. If some domains are found then PLd_{web} score is equal to the sum of Dom_{score} of the corresponding domains.

Hence we have Web_{score}, a score depicting the maliciousness of an HTTP packet and described in Equation 3.

$$Web_{score} = \alpha * D_{web} + \beta * H_{web} + \gamma * PLh_{web} + \delta * PLd_{web} \quad (3)$$

C. Applications

We showed with three metrics, *i.e.* Dom_{score}, $Flow_{score}$ and Web_{score}, how to evaluate the likelihood for a domain name, a flow and respectively an HTTP packet to be malicious. These scores can be sequentially used while a communication is taking place to either raise an alert in an IDS or to block the communication by an IPS.

Figure 2 depicts an example of the information flow while a user performs a HTTP request for the domain *www.malicious.com*. This action is divided in six sequential steps in Figure 2, which also shows how the scores introduced previously are computed. We assume that the honeypot was already running and gathered information.

To request *www.malicious.com* via HTTP, the domain must first be resolved by the recursive DNS server. While the recursive resolution is performed, DNS replies coming from authoritative servers are probed and stored in the database to allow computation of BL_{dom}, C_{dom}, H_{dom} and thus Dom_{score}. Then, the DNS reply is sent to the host which made the original HTTP request. This request is forwarded to the HTTP proxy, then D_{web} and H_{web} relying on the URI and the IP destination are computed as well as a partial Web_{score}. While the HTTP proxy performs the HTTP request, IP flows are exported by the router. H_{flow}, MD_{flow} and NO_{flow} are computed as well as $Flow_{score}$. Finally when the HTTP reply arrives, its payload is analysed to compute PLh_{web} and PLd_{web} in order to deliver a final Web_{score}.

By fixing thresholds to individual scores (BL_{dom}, C_{dom}, H_{dom}, etc.) or to meta scores (Dom_{score}, $Flow_{score}$ and Web_{score}), alerts are raised and connection can be blocked at different steps of a communication. Two thresholds can be fixed for each score, one dedicated to intrusion *detection*: th_d and an other to intrusion *prevention*: th_p with $th_d < th_p$. Properties can be combined in order to wait for having enough information before raising an alert. Typically DNS information is the first we get with our system. Then rules such as the following one can be defined:

```
if $Dom_{score} > th_1$ :
    raise an alert
else if $Dom_{score} > th_2$ & $Web_{score} > th_3$ :
    raise an alert
                                        with $th_2 < th_1$
```

If there is a high confidence with the maliciousness of a domain *i.e.* high value of Dom_{score}, an alert is raised right after Dom_{score} computation. If the score is lower, then the system waits for more information such as information about HTTP communication. Once this information is available Web_{score} is computed and if rules are met an alert is then raised.

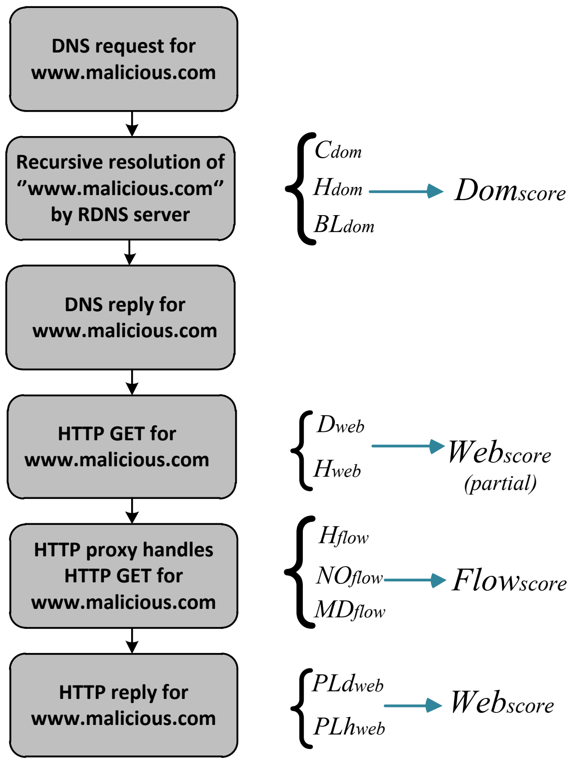
DNS request for www.malicious.com
Recursive resolution of "www.malicious.com" by RDNS server
DNS reply for www.malicious.com
HTTP GET for www.malicious.com
HTTP proxy handles HTTP GET for www.malicious.com
HTTP reply for www.malicious.com
Cdom
Hdom
BLdom
Domscore
Dweb
Hweb
Webscore (partial)
Hflow
NOflow
MDflow
Flowscore
PLdweb
PLhweb
Webscore

Fig. 2. Information flow graph for HTTP request to *www.malicious.com*

Even if first tailored for intrusion detection and prevention, the proposed system can be used for forensic purposes as well. This system stores in a distributed architecture every requested domain name, every flow that goes through the core router of the network and every connection attempts to the honeypot. The distributed data storage architecture along with a big data framework such as Hadoop or Spark provides fast access to worthy historical data. The latter can be used to disclose proof of intrusions as well as their origins long time after these happen.

Some other relevant information can be revealed by our mining system such as finding the machines on the Internet that are the most contacted by a single machine of the enterprise network. This can reveal anomalies if some machines that connect to the honeypots have a lot of interaction with production machines. The number of different protocols and ports used by a single host on the Internet while it is communicating with the enterprise network can be revealed. If a host is not communicating intensively with network machines but always tries new way, by new protocols and new ports, it is suspicious and can disclose a probing campaign. These are few examples of other applications that the proposed architecture can have.

III. Data Storage and Processing

We briefly introduced in previous section that the data exploited by our system is captured at different points and stored in various data storage systems (see Figure 1). This data storage heterogeneity is due to specific requirements for each kind of data (passive DNS, Dionaea, NetFlow). During experiments our system was deployed in a 200 employees company having activities related to electronic payment. To give an idea of the amount of data collected per time period and justify the storage choices here are some values measured during the testing phase:

- Dionaea honeypot: around 1,000 connection attempts from 15 different hosts per day.
- NetFlow: average of 9,600,000 flows per day with an export every minute (approximately 450 Megabytes).
- DNS: around 13 millions DNS replies per day (approximately 1.5 Gigabyte).

Dionaea honeypot, with few daily connections initiated by few attackers does not have high requirements. In addition over the observation period some IP addresses seemed to be redundantly connecting over time. A basic SQLite database is chosen to store the logged information (IP, port, protocol, uploaded payload, etc.). Consultation of the SQLite database is fast enough for this small quantity of data. It is worth noting that even with a larger network to monitor this amount of data would not vary a lot. Having a network of 500 machines or 10,000 would not impact the quantity of data logged by a single honeypot machine deployed in the network.

Storing NetFlow records is more challenging especially if we need the approach to be scalable. NetFlow are exported using `nfdump` every minutes in order to have an almost real time view of the communications. NetFlow records are stored in `nfcapd` format binary files on several distributed servers. The use of several servers is not mandatory for our example but the quantity of flows exported by a router grows with the size of the network it serves. This choice for Netflow storage ensures to meet storage scalability requirements for any network size. However it raises some issues for data treatment as `nfcapd` files are binary files. Big data framework such as Hadoop have an input format for MapReduce tasks that is usually text based. Even though Hadoop supports built-in sequence file format for binary input/output, `nfcapd` files would have to be first converted in a HDFS-specific (Hadoop Distributed File System) sequence file before being uploaded. This process implies a high computational over-head which is time consuming and not acceptable for real-time security monitoring. The alternative is to develop a new API that directly reads NetFlow data in the native `nfcapd` format. Such solution is proposed in [17] through a binary input format for reading packet and NetFlow records concurrently in HDFS. This solution outperforms the previously cited one and is proved fast, providing a throughput of 14 Gbps in a 200-node testbed according to experiments performed in [17].

As for NetFlow, DNS monitoring produces a massive amount of data as seen in our measurements. In addition this volume of data grows with the number of users/machines making DNS queries, *i.e.* the size of the network. Contrarily to NetFlow, all data does not need to be stored for DNS monitoring and only partial information is extracted from DNS packets in order to avoid information redundancy and save storage space. Typically, stored information consists in all possible DNS resource records for a domain name, the TTL of each, some flags, the timestamp for first seen and last seen, etc. All DNS packets do not need to be saved, hence we chose to store DNS related information in a database.

To meet data storage and availability requirements, DNS data is extracted from packets and stored in an Apache Cassandra database. Cassandra [13] is a distributed database solution for data storage that exhibits high performance in data access. It is a decentralized database allowing to store Terabytes of data. In addition, Apache Cassandra integrates Hadoop management since version 0.6 ensuring easy interfacing with state of the art solution for big data processing. This implementation ensures our architecture to fit to larger network than the one we performed tests on.

IV. Performance Analysis

We presented in previous sections an architecture for large scale monitoring. We described data extraction and storage as well as theoretical correlation scheme and their applications. In this section we test the proposed correlation schemes against several big data management systems in order to find the most suitable for such applications.

Two well known open source big data frameworks are assessed, the popular Apache Hadoop ecosystem [18] and the Spark project [19] from AMP Lap of Berkeley university. We focus on performance comparisons of five components of these two frameworks namely Hadoop, Pig, Hive, Shark and Spark.

A. Big Data Tools Presentation

We briefly present here the big data tools we used for the performance assessment of our architecture. For a more detailed description of these tools the reader is referred to [18], [20], [21], [22], [19], [23].

- **Hadoop** [18] is a distributed batch processing framework to process and to analyse large scale datasets. It consists of two primary components, which are HDFS (Hadoop File System) and MapReduce data processing model [20]. Hadoop employs a master/slave architecture to manage a cluster.
- **Hive** [21] is an open source data warehouse infrastructure running on top of Hadoop. It proposes a high level programming language that abstract the implementation of MapReduce jobs to give an user-friendly interface to Hadoop. Commands are expressed in the form of SQL-like queries, thanks to a language called HiveQL.
- **Pig** [22] is also a high level distributed programming model built on top of Hadoop. The main difference between Hive and Pig is on their purpose. Hive is appropriate for database users while Pig targets experienced programmers who are not used to write declarative SQL query.
- **Spark**: Like Hadoop does, Spark [19] proposes a distributed data processing solution for data-intensive applications with the difference that data to process is stored in-memory. Spark has been proved up to 100 times faster than Hadoop for specific tasks like iterative jobs.
- **Shark** [23] is a sub-project of Spark that implements Hadoop's Hive on top of Spark such that it is fully compatible with Hive.

B. Experiments and Results

The five big data solutions are tested in four different scenarios relevant for the computation of the metrics introduced in Section II-B. Experiments were conducted on a cluster of eight machines (one master node and seven slave nodes). Each machine runs a 12.04.4 x86 Ubuntu operating system on an Intel(R) Core(TM)2 Duo with 4GB of RAM. The versions of experimental frameworks are Hadoop-1.2.1, Hive-0.9.0, Pig-0.11.1, Spark-0.6.1 and Shark-0.2.1. The Spark framework was assigned with 2 GB of memory per node *i.e.* 14 GB of working memory in total. The dataset used consists in 767 MB of network traffic. All the scenarios were run ten times for each of the five framework.

The four scenarios are the followings:

- Scenario 1: find packets that match a given source IP address and a given source port.
- Scenario 2: find packets containing a given substring in their payload.
- Scenario 3: count the number of destination IP per source IP and order the result.
- Scenario 4: join two sets according to a common key *i.e.* the source IP addresses.

Scenario 1 corresponds to finding information according to two keys that are a given IP address and a given port. This is exactly what has to be done to compute H_{dom} *i.e.* find whether an IP address associated with a domain has been logged in the honeypot. The same operation is performed to compute H_{flow} and H_{web}, equally BL_{dom} requires the same kind of operation. The time required to perform this operation is given in Table I for each of the five frameworks. This table shows the average, median, minimum and maximum duration taken to perform the task according to the ten tests performed. We can see in the table given results for the scenario 1 that Hadoop and Hive exhibit approximatively the same performance taking around 17.5 seconds to perform it. Pig is a bit slower than the previous ones and the slowest of the five frameworks with 26.475 seconds on average. Finally Spark is much faster than Hadoop doing this task in approximately 1 second. Shark even improves this performance by needing 0.73 seconds on average, it is however less steady than other solutions with sparse amount of time taken among the ten experiments (minimum duration of 0.559 seconds but maximum of 1.648 seconds). For this scenario, Spark seems to be the best candidate exhibiting high speed and steadiness. We mean by steadiness the property to spend approximatively the same amount of time to perform the same task while repeating this task several times. In terms of results it means to have a low standard deviation.

	avg	σ	min	max
Hadoop	17.828	17.836	17.255	18.555
Hive	17.611	17.611	17.565	17.802
Pig	26.475	26.53	25.734	30.792
Spark	1.014	1.017	0.941	1.233
Shark	0.73	0.777	0.559	1.648

Scenario 1

	avg	σ	min	max
Hadoop	17.382	17.385	17.26	18.641
Hive	11.998	13.034	9.649	26.855
Pig	25.762	25.762	25.725	25.834
Spark	0.634	0.636	0.592	0.775
Shark	0.431	0.433	0.4	0.543

Scenario 2

	avg	σ	min	max
Hadoop	34.771	34.773	34.538	35.955
Hive	32.655	32.658	32.023	33.291
Pig	82.744	82.763	82.075	89.477
Spark	1.45	1.979	0.9	6.435
Shark	1.329	1.645	0.913	4.901

Scenario 3

TABLE I
PERFORMANCE OF THE FIVE FRAMEWORKS FOR SCENARIOS 1, 2 & 3 (AVERAGE, MEDIAN, MINIMUM AND MAXIMUM TIME IN SECONDS)

Table I gives the same results for scenarios 2 and 3 as well. Scenario 2 consists in finding a substring in a payload. This operation is performed to compute PLh_{web} and PLd_{web}. This computation requires to mine the content of an HTTP packet to find given code and domain names. It is used as well to compute D_{web} by searching for malicious domains into URIs. Scenario 3 is useful to highlight the machines inside the company network that are communicating the most with machines from the Internet. This is a valuable information for security monitoring and we described this scenario as additional application for our architecture in Section II-C.

Almost the same trends as for scenario 1 is observed for these two other scenarios. Pig is still the slowest solution especially for scenario 3 where it takes 82.744 seconds to perform the task. This is more than twice the time that Hadoop takes and more than a 50-fold increase compared to Spark. Hadoop and Hive perform almost the same except in scenario 2, where Hive is faster than Hadoop on average (12 seconds/17.38 seconds) but Hadoop is steadier than Hive taking always the same amount of time. Spark and Shark again are order of magnitudes faster than the three others, exhibiting results in the order of one seconds for every scenario. Shark still outperforms Spark by some milliseconds. Both are steady for scenario 2 as Hadoop, Hive and Pig are but this does not hold for scenario 3 where there is up to a six-fold fold difference between minimum and maximum time for Spark and Shark.

Finally, scenario 4 is aimed at joining two data feeds according to a common field. The experiment consists in, given a list of source IP addresses, return all packets in our sample that have these IPs as source addresses. The list can be typically a list of malicious IP addresses in order to find all packets coming from these IP addresses. This action has to be done to compute MD_{flow} and NO_{flow} to find if there is common IP address in the DNS database and in NetFlow records. If the resulted set is empty then NO_{flow} is set to 1, otherwise MD_{flow} is computed by adding the corresponding Dom_{score} as described in Section II-B. This join is performed for a list of 10, 20, 30, 40, 50 and $\frac{n}{10}$ IP addresses in the list, where n stands for the total number of distinct IP addresses present in the network traffic sample. Results are presented in Table II with average, median, minimum and maximum time for the five frameworks.

	Hadoop				Hive			
	avg	σ	min	max	avg	σ	min	max
10	17.47	17.47	17.28	18.40	28.71	28.72	28.09	30.65
20	17.49	17.49	17.26	18.31	28.43	28.43	28.0	29.50
30	17.80	17.81	17.25	19.29	28.52	28.52	27.33	29.50
40	17.50	17.50	17.26	18.29	28.44	28.44	28.05	29.35
50	17.83	17.85	17.26	19.26	28.55	28.55	28.03	30.73
$\frac{n}{10}$	17.98	17.98	17.25	18.55	28.30	28.30	27.98	29.17

	avg	σ	min	max
10	67.516	67.527	66.979	72.087
20	67.507	67.518	67.014	72.041
30	67.625	67.636	67.135	72.289
40	68.535	68.563	67.453	72.528
50	68.385	68.408	67.552	72.852
$\frac{n}{10}$	69.048	69.078	67.921	73.259

Pig

	Spark				Shark			
	avg	σ	min	max	avg	σ	min	max
10	1.87	2.27	1.39	6.67	2.78	2.78	2.62	3.16
20	1.43	1.44	1.35	1.79	2.70	2.70	2.57	2.96
30	1.37	1.37	1.33	1.44	2.67	2.67	2.54	2.79
40	1.38	1.38	1.28	1.60	2.64	2.64	2.51	2.92
50	1.33	1.33	1.27	1.52	2.66	2.66	2.52	2.87
$\frac{n}{10}$	1.33	1.33	1.27	1.49	2.65	2.65	2.56	2.80

TABLE II
PERFORMANCE OF THE FIVE FRAMEWORKS FOR SCENARIO 4 (AVERAGE, MEDIAN, MINIMUM AND MAXIMUM TIME IN SECONDS)

For this scenario Spark is the best performer with less than 2 seconds on average, whereas in previous ones it was Shark. Hive, Spark and Shark takes decreasing amount of time to perform this task as the number of values to be joined increases, contrarily to Hadoop and Pig. Spark and Shark are significantly faster than Hadoop, Hive and Pig. All the frameworks are quite steady for this scenario except Spark for a small list of 10 IPs to join.

We showed in this section that Spark and Shark are the best candidates to implement the big data security monitoring system exhibiting fast and steady results in data mining operations. Most of the scores to compute for our correlation schemes (BL_{dom}, H_{dom}, H_{flow}, MD_{flow}, NO_{flow}, D_{web}, H_{web}, PLh_{web}, PLd_{web}) rely on the four scenarios described before and we showed that Spark and Shark are between 10 and 20 times faster than Hadoop for these. We confirm as well results presented in [19], [23] that Shark outperforms Hadoop by 10x.

V. Related Work

Snort [3], Bro [4] and Suricata [5] are centralized IDSs that inspect network packets on the fly to detect intrusions. The proposed system correlates data coming from different network points and several network components.

In the realm of network intrusion detection, the concept of event correlation has been widely explored. The principle is to correlate and establish relations between alarms generated by sensors in order to find similarity in attacks. In [24], Valdes *et al.* correlate alerts based on similarities of their attributes. Alerts are clustered in [25] based on their triggering events in order to identify similar attacks raising different alerts. Alarms from several IDS are correlated in a single system in [26] to achieve high-level description of attacks. We aim in this paper at data correlation more than event correlation. We identify links between malicious domains, malicious URIs, malicious IP address and finally malicious communications in order to disclose common malicious structure. This way DNS requests for malicious domains can be correlated with a-priori benign IP addresses and flows that are actually malicious. Data correlation in this way is briefly introduced in [27] where IP address, traffic routes and commands entered by attackers are correlated.

A service model for network security application is presented in [28]. Ridcciulli proposes an architecture including honeypots that gather cyber-security intelligence. These honeypots continuously gather and record information such as flow data, payloads and signature alert about attackers. This data is stored in a cloud architecture where clients around the world mine and correlate it with monitored flow data in order to detect intrusions. The correlation between honeypot data and clients' data is made with a Google's page ranking like algorithm to find similarities. This work is the closest to ours since it correlates honeypot data and IP flow records in order to disclose intrusions; however it does not include DNS data and HTTP traffic. Due to relying on IP flows and honeypot data, this system has a prominent role of intrusion detection and forensic analysis whereas our system is mainly dedicated to intrusion prevention. Moreover the data management of this system is not described, when we use a scalable distributed data management system leveraging state of the art big data management system in our architecture.

Some other work try to address scalable network intrusion detection [29], [30], [31], [32]. In [29] and [30] stateful analysis of network traffic is performed. Both solutions take advantage of multiple GPUs and multi-core CPUs to parallelize traffic processing and content inspection in order to process several Gigabit of data per second. In [29], every operation is parallelized and mapped to the appropriate device such that the IDS has no serialized component, providing high performance gain. Both solutions [29], [30] are based on Snort intrusion detection system [3]. Another similar solution based on Suricata NIDS [5] is proposed in [31], which also proposes new load balancing. This solution presents scalability both in terms of traffic load and size of the ruleset *i.e.* lot of rules can be set and processed by the IDS thank to this solution. These papers address the same problem as we do: to propose a scalable intrusion detection technique. However, these solutions are based on existing stateful intrusion detection system (Snort or Suricata) that perform in-depth traffic analysis. Their contributions rely on optimizing existing method for network traffic analysis by fully exploiting computers capacity. Contrarily we propose new schemes for network intrusion detection that rely on a subset of the whole network traffic..

Moreover, the heterogeneous distributed data system storage and management system we propose relies on existing well-known solution such as Hadoop [18] or Spark [19] that we compared in different scenarios related to security monitoring. More fundamental performance comparison was described in [33] between the MapReduce paradigm and parallel SQL database management system. Blanas *et al.* analyse the performance of several join algorithms with MapReduce in [34]. Our work is different as we compare existing framework relying on close concepts to address concrete needs of security monitoring on specific data.

VI. Conclusion and Future Work

In this paper we introduced a new scalable architecture for protecting from and detecting network intrusions. This system collects and stores in a distributive manner honeypot data, DNS data, HTTP traffic and IP-flow records. Several correlation schemes relying on this data are introduced and their application, ranging from intrusion detection to forensic analysis, are listed. Five state of the art big data frameworks that can fit for such an architecture are evaluated in four scenarios of data correlation relevant for security monitoring. Out of this performance analysis Spark and Shark appear to be the best performers in all scenarios and thus the best suited to implement the solution.

Even though our architecture computes score with few delay, it still use off-line analysis tool with Hadoop and Shark. Future work will consist in implementing the same system with on-line analysis big data framework such as Spark Streaming [35] or Storm [36].

Acknowledgement

This work is supported by the Fonds National de la Recherche, Luxembourg (Project ID: 3967419).

References

[1] J. P. Anderson, "Computer security threat monitoring and surveillance," Fort Washington, Pennsylvania, Tech. Rep., 1980.

[2] D. E. Denning, "An intrusion-detection model," *IEEE Transactions in Software Engineering*, vol. 13, no. 2, Feb. 1987.

[3] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX conference on System administration*, ser. LISA '99, 1999, pp. 229–238.

[4] V. Paxson, "Bro: a system for detecting network intruders in real-time," in *Proceedings of the 7th conference on USENIX Security Symposium - Volume 7*, ser. SSYM'98, 1998.

[5] "Suricata, open source ids/ips/nsm engine." [Online]. Available: http://www.suricata-ids.org

[6] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for DNS," in *Proceedings of the 19th Usenix Security Symposium*, 2010.

[7] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive DNS analysis," in *Proceedings of NDSS*, 2011.

[8] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of ip flow-based intrusion detection," *Communications Surveys Tutorials, IEEE*, vol. 12, no. 3, pp. 343–356, Third 2010.

[9] J. François, S. Wang, R. State, and T. Engel, "Bottrack: Tracking botnets using netflow and pagerank," in *Proceedings of the 10th International IFIP TC 6 Conference on Networking - Volume Part I*, ser. NETWORKING'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 1–14.

[10] C. Kreibich and J. Crowcroft, "Honeycomb: creating intrusion detection signatures using honeypots," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 51–56, Jan. 2004.

[11] F. Weimer, "Passive dns replication," in *Proceedings of the 17th Annual FIRST Conference on Computer Security Incident Handling*, 2005.

[12] R. Edmonds, "ISC Passive DNS Architecture," Internet Systems Consortium, Inc., Tech. Rep., 2012. [Online]. Available: https://security.isc.org/Passive_DNS/passive-dns-architecture.pdf

[13] A. Lakshman and P. Malik, "Cassandra: structured storage system on a p2p network," in *Proceedings of the 28th ACM symposium on Principles of distributed computing*, ser. PODC '09. New York, NY, USA: ACM, 2009, pp. 5–5.

[14] "Cisco netflow." [Online]. Available: http://www.cisco.com/web/go/netflow

[15] "Dionaea, catches bugs." [Online]. Available: http://dionaea.carnivore.it/

[16] S. Marchal, J. François, C. Wagner, R. State, A. Dulaunoy, T. Engel, and O. Festor, "DNSSM: A large scale passive DNS security monitoring framework," ser. NOMS'12, 2012.

[17] Y. Lee and Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 1, pp. 5–13, 2012.

[18] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, June 2009.

[19] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, ser. HotCloud'10, 2010.

[20] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Symposium on Opearting Systems Design & Implementation (OSDI)*. USENIX Association, 2004.

[21] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1626–1629, Aug. 2009.

[22] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 1099–1110.

[23] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: fast data analysis using coarse-grained distributed memory," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12, 2012, pp. 689–692.

[24] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, 2001, pp. 54–68.

[25] D. Xu and P. Ning, "Alert correlation through triggering events and common resources," in *Proceedings of the 20th Annual Computer Security Applications Conference*, 2004, pp. 360–369.

[26] T. Chyssler, S. Burschka, M. Semling, T. Lingvall, and K. Burbeck, "Alarm reduction and correlation in intrusion detection systems," in *Proceedings of Detection of Intrusions and Malware Vulnerability Assessment workshop (DIMVA)*, 2004, pp. 9–24.

[27] C. Endorf, E. Schultz, and J. Mellander, *Intrusion detection & prevention*. McGraw-Hill, 2004.

[28] L. Ricciulli, "A service model for network security applications," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW '10, 2010.

[29] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "Midea: A multi-parallel intrusion detection architecture," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 297–308.

[30] M. A. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park, "Kargus: A highly-scalable software-based intrusion detection system," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 317–328.

[31] H. Jiang, G. Zhang, G. Xie, K. Salamatian, and L. Mathy, "Scalable high-performance parallel design for network intrusion detection systems on many-core processors," in *Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 137–146.

[32] H. Gill, D. Lin, X. Han, C. Nguyen, T. Gill, and B. T. Loo, "Scalanytics: A declarative multi-core platform for scalable composable traffic analytics," in *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '13. New York, NY, USA: ACM, 2013, pp. 61–72.

[33] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 165–178.

[34] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, "A comparison of join algorithms for log processing in mapreduce," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 975–986.

[35] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "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 Ccomputing*, ser. HotCloud'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 10–10.

[36] "Storm." [Online]. Available: http://storm-project.net/